m3-mutex

Выпуск

BARS Group

17 March 2014

Contents:

Глава 1

Общее описание

1.1 Назначение модуля и основной функционал

Модуль предназначен для работы с семафорами и глобальными блокировками операций.

1.2 Состав и структура

1.2.1 Диаграмма классов

1.2.2 Структура файлов

api.py

Данный файл содержит следующие api-функции:

get_backend(mutex_id) *Получает бекэнд*

param mutex_id семафор

type mutex_id объект класса domain.Mutex или domain.MutexID

return объект класса ModelMutexBackend

capture_mutex(mutex_id, owner=None, auto_release=TimeoutAutoRelease(timeout=300), status_data=
Устанавливает семафор с таймаутом в 300 секунд (5 минут) по умолчанию

param mutex_id семафор

type mutex_id объект класса domain.Mutex или domain.MutexID

param owner владелец семафора. Если передать значение None, то будет использовано значение из thread-locals

type owner объект класса domain.MutexOwner

param auto_release правило автоосвобождения данного семаформа. По умолчанию используется автоматическое освобождение по таймауту в 5 минут

param str status_data статусная информация

return объект класса domain.Mutex

release_mutex(mutex_id, owner=None) *Освобождает семафор*

param mutex_id семафор

type mutex_id объект класса domain.Mutex или domain.MutexID

param owner владелец семафора. Если передать значение None, то будет использовано значение из thread-locals

type owner объект класса domain.MutexOwner

return

request_mutex(mutex_id, owner=None) *Проверяет, свободен ли семафор*

param mutex_id семафор

type mutex_id объект класса domain.Mutex или domain.MutexID

param owner владелец семафора. Если передать значение None, то будет использовано значение из thread-locals

type owner объект класса domain.MutexOwner

return кортеж из состояния семафора MutexState и семафора или None

get_mutex_list(mutex_query=MutexQuery()) *Возвращает список семафоров*

param mutex_query условия отбора семафоров

type mutex_query объект класса MutexQuery

return список семафоров

backends.py

Данный файл содержит классы бекэндов для управления семафорами:

class BaseMutexBackend(object) *Базовый класс (интерфейс) бекэнда для управления семафорами.*

Содержит следующие методы:

capture_mutex(self, mutex_id, owner=None, auto_release=TimeoutAutoRelease(), status_data= *Метод захвата семафора.*

param mutex_id семафор

type mutex_id объект класса domain.Mutex или domain.MutexID

param owner владелец семафора. Если передать значение None, то будет использовано значение из thread-locals

type owner объект класса domain.MutexOwner

param auto_release правило автоосвобождения данного семаформа. По умолчанию используется автоматическое освобождение по таймауту в 5 минут.

param str status_data статусная информация

return объект класса domain.Mutex

raise MutexBusy

release_mutex(self, mutex_id, owner=None) *Метод освобождения семафора. В случае, если семафор был ранее захвачен не нами.*

param mutex_id семафор

type mutex_id объект класса domain.Mutex или domain.MutexID

param owner владелец семафора. Если передать значение None, то будет использовано значение из thread-locals

type owner объект класса domain.MutexOwner

return

raise MutexBusy

request_mutex(self, mutex_id, owner=None) *Метод проверки состояния семафора.*

param mutex_id семафор

type mutex_id объект класса domain.Mutex или domain.MutexID

param owner владелец семафора. Если передать значение None, то будет использовано значение из thread-locals

type owner объект класса domain.MutexOwner

return кортеж из состояния семафора MutexState и семафора или None

_read_mutex(self, mutex_id) *Внутренний метод чтения информации о семафоре из хранилища*

param mutex_id семафор

type mutex_id объект класса domain.Mutex или domain.MutexID

_add_mutex(self, mutex) *Внутренний метод добавления семафора в хранилище*

param mutex семафор

type mutex объект класса domain.Mutex

_remove_mutex(self, mutex_id) *Внутренний метод удаления семафора из хранилища*

param mutex_id семафор

type mutex_id объект класса domain.Mutex или domain.MutexID

class ModelMutexBackend(BaseMutexBackend) *Бекэнд, который реализует хранение семафоров в базе данных*

class SessionMutexBackend(BaseMutexBackend) *Бекэнд, который реализует хранение семафоров в текущей обработки запросов. Бекэнд необходимо использовать для выставления только в рамках обработки текущего запроса.*

exceptions.py

Содержит исключения, возникающие внутри модуля

class MutexBusy(Exception) *Исключительная ситуация, возникающая при попытке захвата семафора, который уже захвачен другим владельцем*

helpers.py

Содержит вспомогательные функции

compare_owners(owner1, owner2, soft=False)

Производит сравнение объектов двух владельцев семафоров. Возвращает True в случае, если владельцы идентичны. :param owner1: первый владелец :param owner2: второй владелец :param soft:

return bool

get_default_owner() *Возвращает объект MutexOwner, который представляет владельца семафоров в текущей сессии обработки запроса. Информация о текущем владельце читается из thread-locals. В случае, если информация о текущей сессии в thread-locals отсутствует, то считается, что работа с системой производится из shell-консоли и параметры владельца заполняются на основании констант* **CONSOLE_SESSION_KEY**, **CONSOLE_USER_ID** *и* **CONSOLE_USER_NAME**

return объект класса MutexOwner

get_session_info() *Возвращает информацию о текущей сессии обработки информации. В случае если работа с системой происходит из shell-консоли, то возвращается кортеж **CONSOLE_SESSION_KEY*.**

return tuple(ключ сессии, идентификатор пользователя)

get_backend(mutex_id) *Возвращает backend, который используется для хранения информации о семафорах.*

param mutex_id идентификатор семафора, для которого определяется backend

models.py

Содержит Django-модели модуля

class MutexModel(models.Model) *Модель хранения информации о семафоре*

Содержит поля:

- **mutex_group** - группа семафора
- **mutex_mode** - тип семафора
- **mutex_id** - id семафора
- **owner_session** - сессия владельца семафора
- **owner_host** - хост владельца семафора
- **owner_id** - id владельца семафора
- **owner_login** - логин владельца семафора
- **owner_name** - имя владельца семафора

- **auto_release_rule** - правило автоосвобождения семафора
- **auto_release_config** - настройка автоосвобождения семафора
- **captured_since** - дата и время захвата семафора
- **status_data** - информация о статусе семафора

domain.py

Содержит основные и вспомогательные классы модуля

class MutexID(object) *Инкапсуляция над идентификатором экземпляра семафора*

class MutexOwner(object) *Инкапсуляция над владельцем семафора*

Содержит следующие методы:

def __init__(self, session_id='', user_id=0, name='', login='', host='')

param session_id идентификатор сессии

param name наименование (например, ФИО) владельца

param user_id уникальный идентификатор владельца

param login логин пользователя

param host хост, с которого был выставлен семафор

class SystemOwner(MutexOwner) *Владелец, представленный в виде системного процесса*

class MutexState *Класс-перечисление возможных состояний семафора*

Содержит состояния:

- **FREE** - семафор свободен
- **CAPTURED_BY_ME** - семафор захвачен нами
- **CAPTURED_BY_OTHER** - семафор захвачен не нами

class Mutex(object) *Класс семафора*

Содержит следующие методы:

__init__(self, id=MutexID(), owner=MutexOwner(), auto_release=None)

param id id семафора

param owner владелец семафора

param auto_release правило автоосвобождения семафора

check_owner(self, owner) *Возвращает True в случае, если указанный в параметрах owner совпадает с владельцем семафора*

param owner владелец

return bool

class MutexAutoReleaseRule(object) *Базовый класс, устанавливающий правила автоматического освобождения(снятия) семафора. Данный механизм необходим для того, чтобы семафоры не оставались в системе "навсегда"*

Содержит следующие методы:

check(self, mutex) *Основной метод, который возвращает True в случае, если указанный данный семафор может быть освобожден в автоматическом режиме*

param mutex семафор

type mutex объект класса domain.Mutex

dump(self) *Возвращает кортеж из двух элементов для сохранения алгоритма автоматического освобождения семафоров в текстовом виде. Данный метод, будучи переопределенным в дочерних классах, должен вернуть кортеж из двух элементов ('код правила', 'упакованные параметры срабатывания правила')*

restore(self, config) *Читает информацию о конфигурации условий автаматического освобождения семафоров из текстовой строки.*

param config конфигурация

get_rule_class(str='timeout') *Статический метод. Получает класс правила автоосвобождения*

param str наименование правила

class TimeoutAutoRelease(MutexAutoReleaseRule) *Класс, реализующий освобождение семафора на основании превышения времени ожидания.*

class MutexQuery(object) *Класс, представляющий запрос на получение информации*

Содержит следующие методы:

__init__(self, filter='', start=0, offset=-1)

param filter правила фильтрации

param start индекс начала

param offset прерращение

1.3 Лицензия

Глава 2

Установка

2.1 Необходимые библиотеки и другие модули

Для работы m3-mutex необходимы следующие модули:

- django
- south
- m3_legacy

2.2 Установка и настройка

Для подключения m3-mutex к django-проекту необходимо добавить в список установленных приложений:

```
INSTALLED_APPS = {
        ...,
        'm3_mutex',
        ...
}
```

Глава 3

Простое использование

3.1 Простые примеры использования модуля

В качестве примера работы с m3-mutex реализуем небольшой класс-семафор, который будет блокировать повторное редактирование уже заданного объекта:

```
class ObjEditMutex(object):
u"""Класс-семафор, блокирующий повторное редактирование объекта
"""
```

Добавим метод, который позволит нам получать id объекта:

```
def get_obj_id(self, obj_id):
    u"""Подмена obj_id, для возможности переопределения класса.
    Позволяет использовать свой уникальный obj_id
    без изменения логики родительского класса.
    """
    return obj_id
```

Добавим метод, который позволит нам получать семафор по типу и id объекта:

```
def get_mutex_id(self, mode, obj_id):
    u"""Возвращает id семафора для указанного типа и id объекта

    :param basestring mode: Тип объекта
    :param int obj_id: id объекта

    :rtype: int
    """
    return m3_mutex.MutexID(mode=mode, id=str(obj_id))
```

Добавим метод, который позволит нам проверять наличие семафора:

```
def check(self, obj_id, mode, request_id):
    u"""Проверка на наличие семафора.
    Возвращает True, если объект не заблокирован или заблокирован нами.
    В противном случае возвращает False
    """
    obj_id = self.get_obj_id(obj_id)
    mutex_id = self.get_mutex_id(mode=mode, obj_id=obj_id)

    # По умолчанию семафор занят
    is_free = False
```

```
    state, mt = m3_mutex.request_mutex(mutex_id)

    # Проверяем свободен ли семафор
    if state == m3_mutex.MutexState.FREE:
        is_free = True
    # Если семафорт занят нами в текущем запросе, то тоже все ОК
    elif state == m3_mutex.MutexState.CAPTURED_BY_ME:
        is_free = True

    return is_free
```

Добавим метод, который позволит нам создавать семафор для последующей его блокировки:

```
def block(self, obj_id, mode, request_id):
    u"""Создание семафора и его блокировка.
    Передаётся id объекта, тип объекта и id запроса.

    :param int obj_id: id объекта
    :param basestring mode: тип объекта
    :param int request_id: id запроса

    :rtype: basestring or None
    """
    blocker = None
    obj_id = self.get_obj_id(obj_id)
    mutex_id = self.get_mutex_id(mode=mode, obj_id=obj_id)

    try:
        # Получаем состояние
        state, mt = m3_mutex.request_mutex(mutex_id)

        # Если mutex никем не занят, в том числе нами из другого места
        if state == m3_mutex.MutexState.FREE:
            # Захватываем mutex
            m3_mutex.capture_mutex(mutex_id, status_data=request_id)

        # Иначе блокируем
        else:
            # Получаем владельца mutex
            blocker = mt.owner.name

    # Если занят
    except m3_mutex.MutexBusy:
        blocker = u'Блокировщик неизвестен'

    return blocker
```

Добавим метод, который позволит нам обновлять блокировку семафора:

```
def refresh(self, obj_id, mode, request_id):
    u"""Обновление блокировки семафора, чтобы он не освободился по тайм-ауту.
    Передаётся id объекта, тип объекта и id запроса.

    :param int obj_id: id объекта
    :param basestring mode: тип объекта
    :param int request_id: id запроса

    :rtype: basestring or None
    """
```

```
    obj_id = self.get_obj_id(obj_id)
    mutex_id = self.get_mutex_id(mode=mode, obj_id=obj_id)

    # Получаем состояние
    state, mt = m3_mutex.request_mutex(mutex_id)

    # Если заблокирован нами и не из другого запроса
    if (state == m3_mutex.MutexState.CAPTURED_BY_ME
            and mt.status_data == request_id):
        # Обновляем блокировку для повторного захвата
        m3_mutex.capture_mutex(mutex_id, status_data=request_id)
    else:
        return u"Блокировка объекта снята!"

    return None
```

Добавим метод, который позволит нам освобождать семафор:

```
def release(self, obj_id, mode, request_id):
    u"""Освобождение семафора.
    Передаётся id объекта, тип объекта и id запроса.
    """
    obj_id = self.get_obj_id(obj_id)
    mutex_id = m3_mutex.MutexID(mode=mode, id=obj_id)

    state, mt = m3_mutex.request_mutex(mutex_id)

    if state == m3_mutex.MutexState.CAPTURED_BY_ME:
        mt = m3_mutex.capture_mutex(mutex_id)
        try:
            if mt.status_data == request_id:
                m3_mutex.release_mutex(mutex_id)
        except m3_mutex.MutexBusy:
            pass
    return None
```

После этого, данный семафор можно использовать следующим образом:

```
# создаем семафор
mutex = ObjEditMutex()
# создаем или получаем каким - либо образом объект
some_obj = SomeObjClass()

# блокируем объект
blocker = mutex.block(some_obj.id, some_obj.type, some_obj.uuid)
# если нам удалось заблокировать объект
if blocker is None:
        # делаем что - то с объектом
        do_something(some_obj)
        # освобождаем объект
        mutex.release(some_obj.id, some_obj.type, some_obj.uuid)
        print 'Операция с объектом завершена'
else:
        # говорим, что объект занят
        print u'Объект занят'
```

Глава 4

Indices and tables

- *genindex*
- *modindex*
- *search*