Object Pool

Release 1.0

Feb 09, 2020

All Contents

Table of contents

CHAPTER 1

Introduction

Object pool library creates a pool of resource class instance and use them in your project. Pool is implemented using python built in library Queue.

Let's say for example, you need multiple firefox browser object in headless mode to be available for client request to process or some testing or scraping.

- Each time creating a new browser instance is time consuming task which will make client to wait.
- If you have one browser instance and manage with browser tab, it will become cumbersome to maintain and debug in case of any issue arises.

Object Pool will help you to manage in that situation as it creates resource pool and provides to each client when it requests. Thus separating the process from one another without waiting or creating new instance on the spot.

New instance creation will happen with ObjectPool but when no resources available in the pool for a client. ObjectPool will not fail in case resource is not available, but it will create one and provide to client. Thus keeping your system/process available at the cost of little performance.

CHAPTER 2

Little about resource

Resource is a class which will be pooled. Resource class can have following properties.

- Resource can have **check_invalid** method to provide custom validation. This method should return boolean values. This will be called while cleaning up of the resource. When this method is not defined, only

```
def check_invalid(self, **stats):
    '''Returns True if the resource is valid, otherwise False'''
    return False
```

- Resource should have **clean_up** method to provide clean up process for resource. Since resource information is not available to object, how a resource should be clean up using this method. It is advised to keep clean up method mandatory as resource such as database connection or browser process will affect system performance.

```
def clean_up(self, **stats):
    self.browser.quit()
    self.browser = None
```

- Resource methods check_invalid and clean_up will have keyword argument stats. Stats will be have below information regarding resource.
  - count - resource usage count.
  - last_used - last usage time of the resource.
  - new - is updated after the time time use or recreated.

Example - Resource class

Let's have an example of Browser pool for clients. We have below browser class. perquisite for this testing are below.

- Selenium
- Geckodriver

```
from selenium.webdriver import Firefox, FirefoxProfile
from selenium.webdriver.firefox.options import Options
import time

class FirefoxBrowser:
    """
    This is browser resource class for ObjectPool. This class demonstrate␣
↪how to create resource class
    and implement methods described in the user guide section.
    """

    def __init__(self):
        self.browser = FirefoxBrowser.create_ff_browser()
        self.page_title = None

    @classmethod
    def create_ff_browser(cls):
        """Returns headless firefox browser object"""
        profile = FirefoxProfile().set_preference("intl.accept_languages",
↪"es")
        opts = Options()
        opts.headless = True
        browser = Firefox(options=opts, firefox_profile=profile)
        return browser

    def get_page_title(self, url):
        """Returns page title of the url"""
        browser = self.browser
        browser.get(url)
        self.page_title = browser.title
        return self.page_title

    def clean_up(self, **stats):
        """quits browser and sets None, when this method is called"""
        self.browser.quit()
        self.browser = None

    def check_invalid(self, **stats):
        """Returns invalid=True if the browser accessed google web page,␣
↪otherwise False"""
        if self.page_title == 'Google':
            return True
        return False
```

CHAPTER 3

Features

Resources to client are provided by ObjectPool. Queueing and cleaning up are automatically taken care by ObjectPool.

- *get()* method used conjunction with **with statement**.
- *get()* removes resource from pool and provides to client.
- when client code exits with statement, utilized resource will be queued in the pool to use after validation.
- get() method provides two object, one is resource and respective stats in the pool.
  - resource - resource class instance. Through this, you can call any method in the class you defined.
  - stats - resource statistics. This will be used for you to perform custom validation in the resource class.

```
from object_pool import ObjectPool
browser_pool = ObjectPool(FirefoxBrowser)

# browser will be created as it is lazy when get() method called.
# browser will be placed in the queue when exits with statement.
with browser_pool.get() as (browser, browser_stats):
    # you can call any method defined in the resource and perform␣
↪operation
    title = browser.get_page_title('https://www.google.co.in/')
```

Once you define a resource class, you can use below features to create the pool for the resource.

3.1 Lazy behaviour

lazy=True Resource pool will be created with zero resource when initiated. New resource will be created when requested and pooled till it reach it's maximum capacity.

```
browser_pool = ObjectPool(FirefoxBrowser, lazy=True)
```

3.2 Unlimited resource

When **max_capacity=0** is set, pool grow unlimited. This option needs to be used with cautious and advised used with expiry options, as resource grows without clean up which will lead to performance issue.

```
browser_pool = ObjectPool(FirefoxBrowser, max_capacity=0)
```

3.3 Resource Expiration

Resource expiration methods can used to set resource expiry for a resource. Object pool refill the pool with new resource.

3.3.1 Limited time resource

When pool is created with **expires=600**, resource will be cleaned up and removed from the pool *after 10 mins*. *expires=0* resource will never expire.

```
# each resource in the pool will expire in 10 mins from the created time.
browser_pool = ObjectPool(FirefoxBrowser, expires=600)
```

3.3.2 Resource usage policy

When **max_reusable=6**, resource can be used only 6 times by any client. After this usage limit, resource will be cleaned up and new resource will be created.

```
# each resource in the pool will expire in 10 mins from the created time
# or 20 times used by clients
browser_pool = ObjectPool(FirefoxBrowser, max_reusable=20, expires=600)
```

3.4 Speed up creation

When **cloning=True**, object_pool will create new resource by cloning reserved resource. This will not fit for all resources. Best It should be tested by user, before using it. By default cloning is disabled.

```
# new resource will be created by cloning reserved resource instance.
browser_pool = ObjectPool(FirefoxBrowser, cloning=True)
```

```
Seleinum browser or db connection resources will not be able to use cloning. But If␣
↪you have
any custom object which performs long running calculation and creates instance,␣
↪cloning will
useful that time.
```

```
Reserved resource is a instance of resource class for cloning to create new resource.
This reserved instance will not be part of the pool.
```

3.5 Custom validation

User's custom validation can be defined in **check_invalid** in the resource class. This will be called while cleaning up of the resource.

Lets say you dont want to access the google web page more than once in the same browser. You can invalidate with **check_invalid** method and clean up the browser as below method.

```
def check_invalid(self, **stats):
    # invalidate browser which accessed google web page to create new resource in␣
↪place.
    if self.page_title == 'Google':
        return True
    return False
```

3.5.1 Dev Guide

Table of contents

The core module of the object pool creation library and contains all the necessary classes.

ObjectPool Class

class object_pool.pool.**ObjectPool**(*klass*, *max_capacity=20*, *min_init=3*, *max_reusable=20*, *expires=600*, *lazy=False*, *pre_check=False*, *post_check=True*, *cloning=False*)

This is singleton object pool class. It creates pool, checks expiry and validation of the resource.

Parameters

- **base_klass** – class for which pool will be created.

  In the below example, Pool will be created for Browser class and used.

```
class Browser:
    def __init__(self):
        self.browser = self.__class__.__create_connection()

    @staticmethod
    def __create_connection():
        obj = "connection_object"
        return obj

    def do_work(self):
        return True
```

(continues on next page)

(continued from previous page)

```
    def clean_up(self, **stats):
        print("close connection object")

    def check_invalid(self, **stats):
        '''Returns True if resource is valid, otherwise False'''
        return False
```

- **`lazy`** – by default, resources are created when initiated. lazy option will skip resource creation on init and will create when the pool item is requested.
- **`min_init`** – minimum resources will be created while initiating.

Note: When **lazy=True** is set, *min* option is not respected.

- **`max_capacity`** – Maximum capacity of the pool. Pool will be created with **min_init** capacity. But it go grow up to max_capacity.

Note:

- *max_capacity* is not a hard constraint to the pool. When the client request for a resource, and no resources are available for client, new resource will be created and provided to the client. But this extra resource will not be queue, it will be cleaned up without performing any validation.
- pool is implemented using Queue. But maxsize is not provided to handle *max_capacity*. This is a implementation choice.
- *As creation and cleaning up of additional resource performed when pool gets full, This will slow down the program. This is a cue to check bottleneck on client processing.*

- **`max_reusable`** – maximum number of times resource can be reused. Once this exceeds, respective resource will be destroyed and new resource will be created. By default, 20 is set. You can disable this by setting to 0.
- **`expires`** – resource expiration in seconds.

Note: **Example:** expires=600, Resource will be only alive for 10 minutes from the creation.

- **`pre_check`** – resource validation is performed before requesting the resource. This is disabled by default.
- **`post_check`** – resource expiration checked after resource is being used. This is the default option.

Note: Base class should define clean_up method when **expiry** is set or **max_reusable** is set or **check_valid** method is defined in the class. If the clean up such as closing database connection or closing browser are not performed, those process will run in the background and cause performance issue in the system.

- **`cloning`** – reserved resource will be created to create new resource, in case of resource expiration. Cloning is disabled by default. You can enable by passing cloning=True.

Note: Reserved resource will be created even **lazy=True** option provided to reduce the resource creation time.

```
>>> class Browser: # Objective pool class
...     def __init__(self):
...         self._browser = "connected!"
...
...     def do_work(self):
...         return "job done!"
...
...     def clean_up(self, **stats):
...         print("stats contains resource stats")
>>> # default ConnectionPool options
>>> connection_pool = ObjectPool(Browser, max_capacity=20, min_
↪init=3, max_reusable=20,
...                 expires=600, lazy=False, pre_check=False, post_
↪check=True, cloning=False)
```

get()

Creates contextmanager instance and returns resource and stats

```
>>> class Connection:
...     def __init__(self):
...         self._conn = "connected!"
...
...     def do_work(self):
...         print("job done!")
...
>>> pool = ObjectPool(Connection, min_init=3)
>>>
>>> with pool.get() as (resource, resource_stats):
...     resource.do_work()
job done!
```

get_pool_size()

Returns the size of the pool (queue).

Any operation on the queue results in different output time to time as the resources are removed from queue and added back.

```
>>> pool = ObjectPool(Connection, min_init=3)
>>> print(pool.get_pool_size())
3
```

static pool_exists(*klass*)

Return True if the pool is already created, False otherwise.

```
>>> pool = ObjectPool(Connection, min_init=3)
>>> ObjectPool.pool_exists(Connection)
True
>>> ObjectPool.pool_exists(DummyConnection)
False
```

destroy()

Removes pool from the registry and performs clean up.

When the pool is destroyed and *cleanup_func* is provided or class has *clean_up* method defined while creating, respective clean up method is called to clean up on the object.

Example: When the connection pool is destroyed, all the connection objects in the pool will be closed if the cleanup method is provided when creating the pool.

```
>>> class Connection:
...     def __init__(self):
...         self._conn = "connected!"
...
...     def do_work(self):
...         return "job done!"
...
...     def clean_up(self, **resource_stats):
...         return "cleanup performed!"
...
>>> pool = ObjectPool(Connection, min_init=1)
>>> pool.destroy()
cleanup performed!
```

is_pool_full()

Return True if the pool is full, False otherwise.

```
>>> pool = ObjectPool(Connection, min_init=3, max_capacity=3)
>>> pool.is_pool_full()
True
```

Note: This method will always return False when **max_capacity=0**, As the pool grow unlimited.

3.5.2 Readme File

ObjectPool

- Explore the docs »
- Source Code »
- Report Bug »
- Request Feature »

Table of contents

Object Pool

Object pool library creates a pool of resource class instance and use them in your project. Pool is implemented using python built in library Queue.

Let's say for example, you need multiple firefox browser object in headless mode to be available for client request to process or some testing or scraping.

- Each time creating a new browser instance is time consuming task which will make client to wait.
- If you have one browser instance and manage with browser tab, it will become cumbersome to maintain and debug in case of any issue arises.

Object Pool will help you to manage in that situation as it creates resource pool and provides to each client when it requests. Thus separating the process from one another without waiting or creating new instance on the spot.

How to install

```
pip install py-object-pool
```

Source

Find the latest version on GitHub - ObjectPool.

Feel free to fork and contribute!

Requirements

Python 3.6 and above

Code Example

Below example will help you to understand how to create resource class for ObjectPool and use them in your project.

In the below example, *Browser* is a resource class and *ff_browser_pool* is a pool of Browser.

Please refer the user guide for more details.

Sample resource class

```
from selenium.webdriver import Firefox, FirefoxProfile
from selenium.webdriver.firefox.options import Options


class FirefoxBrowser:
    """
    This is browser resource class for ObjectPool. This class demonstrate how to␣
↪create resource class
    and implement methods described in the user guide section.
    """
```

(continues on next page)

(continued from previous page)

```
    def __init__(self):
        self.browser = FirefoxBrowser.create_ff_browser()
        self.page_title = None

    @classmethod
    def create_ff_browser(cls):
        """Returns headless firefox browser object"""
        profile = FirefoxProfile().set_preference("intl.accept_languages", "es")
        opts = Options()
        opts.headless = True
        browser = Firefox(options=opts, firefox_profile=profile)
        return browser

    def get_page_title(self, url):
        """Returns page title of the url"""
        browser = self.browser
        browser.get(url)
        self.page_title = browser.title
        return self.page_title

    def clean_up(self, **stats):
        """quits browser and sets None, when this method is called"""
        self.browser.quit()
        self.browser = None

    def check_invalid(self, **stats):
        """Returns invalid=True if the browser accessed google web page, otherwise␣
↪False"""
        if self.page_title == 'Google':
            return True
        return False
```

Sample client block

```
from object_pool import ObjectPool

ff_browser_pool = ObjectPool(FirefoxBrowser, min_init=2)

with ff_browser_pool.get() as (browser, browser_stats):
    title = browser.get_page_title('https://www.google.co.in/')
```

Authors

Durai Pandian - *Initial work* - dduraipandian

License

This project is licensed under the MIT License - see the LICENSE file for details

Python Module Index

o

Index