Python __del__

Summary: in this tutorial, you will learn about the Python `__del__` special method and understand how it works.

Introduction to the Python __del__ method

In Python, the garbage collector (https://www.pythontutorial.net/advanced-python/python-garbage-collection/) manages memory automatically. The garbage collector will destroy the objects that are not referenced (https://www.pythontutorial.net/advanced-python/python-references/) .

If an object implements the `__del__` method, Python calls the `__del__` method right before the garbage collector destroys the object.

However, the garbage collector determines when to destroy the object. Therefore, it determines when the `__del__` method will be called.

The `__del__` is sometimes referred to as a class finalizer. Note that `__del__` is not the destructor because the garbage collector destroys the object, not the `__del__` method.

The Python __del__ pitfalls

Python calls the `__del__` method when all object references are gone. And you cannot control it in most cases.

Therefore, you should not use the `__del__` method to clean up the resources. It's recommended to use the context manager.

If the `__del__` contains references to objects, the garbage collector will also destroy these objects when the `__del__` is called.

If the `__del__` references the global objects, it may create unexpected behaviors.

If an exception occurs inside the `__del__` method, Python does not raise the exception but keeps it silent.

Also, Python sends the exception message to the stderr. Therefore, the main program will be able to be aware of the exceptions during the finalization.

In practice, you'll rarely use the `__del__` method.

Python __del__ example

The following defines a `Person` class (https://www.pythontutorial.net/python-oop/python-class/) with the special `__del__` method, create a new instance of the `Person`, and set it to `None` (https://www.pythontutorial.net/advanced-python/python-none/):

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __del__(self):
        print('__del__ was called')


if __name__ == '__main__':
```

```
person = Person('John Doe', 23)
person = None
```

Output:

```
__del__ was called
```

When we set the `person` object to `None`, the garbage collector destroys it because there is no reference. Therefore, the `__del__` method was called.

If you use the `del` keyword to delete the `person` object, the `__del__` method is also called:

```
person = Person('John Doe', 23)
del person
```

Output:

```
__del__ was called
```

However, the `del` statement doesn't cause a call to the `__del__` method if the object has a reference.

Summary

- Python calls the `__del__` method right before the garbage collector destroys the object.
- The garbage collector destroys an object when there is no reference to the object.
- Exception occurs inside the `__del__` method is not raised but silent.
- Avoid using `__del__` for clean up resources; use the context manager instead.