Заметки об объектной системе языка Python ч.1 из песочницы

Python\*

Несколько заметок об объектной системе python'a. Рассчитаны на тех, кто уже умеет программировать на python. Речь идет только о новых классах (new-style classes) в python 2.3 и выше. В этой статье рассказывается, что такое объекты и как происходит поиск атрибутов.

Объекты

Все данные в питоне — это объекты. Каждый объект имеет 2 специальных атрибута \_\_class\_\_ и \_\_dict\_\_.

\_\_class\_\_ — определяет класс или тип, экзмепляром которого является объект. Тип (или класс объекта) определяет его поведение; он есть у всех объектов, в том числе и встроенных. Тип и класс — это разные названия одного и того же. x.\_\_class\_\_ <==> type(x).

\_\_dict\_\_ словарь, дающий доступ к внутреннему пространству имен, он есть почти у всех объектов, у многих встроенных типов его нет.

Примеры.

>>> def foo(): pass

...

>>> foo.\_\_class\_\_

<type 'function'>

>>> foo.\_\_dict\_\_

{}

>>> (42).\_\_dict\_\_

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'int' object has no attribute '\_\_dict\_\_'

>>> (42).\_\_class\_\_

<type 'int'>

>>> class A(object):

... qux = 'A'

... def \_\_init\_\_(self, name):

... self.name=name

... def foo(self):

... print 'foo'

...

>>> a = A('a')

У a тоже есть \_\_dict\_\_ и \_\_class\_\_:

>>> a.\_\_dict\_\_ {'name': 'a'}

>>> a.\_\_class\_\_

<class '\_\_main\_\_.A'>

>>> type(a)

<class '\_\_main\_\_.A'>

>>> a.\_\_class\_\_ is type(a)

True

Класс и тип — это одно и то же.

>>> a.\_\_class\_\_ is type(a) is A

True

a.\_\_dict\_\_ — это словарь, в котором находятся внутренние (или специфичные для объекта) атрибуты, в данном случае 'name'. А в a.\_\_class\_\_ класс (тип).

И, например, в методах класса присваивание self.foo = bar практически идентично self.\_\_dict\_\_['foo'] = bar или сводится к аналогичному вызову.

В \_\_dict\_\_ объекта нет методов класса, дескрипторов, классовых переменных, свойств, статических методов класса, все они определяются динамически с помощью класса из \_\_class\_\_ атрибута, и являются специфичными именно для класса (типа) объекта, а не для самого объекта.

Пример. Переопределим класс объекта a:

>>> class B(object):

... qux = 'B'

... def \_\_init\_\_(self):

... self.name = 'B object'

... def bar(self):

... print 'bar'

...

>>> a.\_\_dict\_\_

{'name': 'a'}

>>> a.foo()

foo

>>> a.\_\_class\_\_

<class '\_\_main\_\_.A'>

>>> a.\_\_class\_\_ = B

>>> a.\_\_class\_\_

<class '\_\_main\_\_.B'>

Смотрим, что поменялось.

Значение a.name осталось прежним, т.е. \_\_init\_\_ не вызывался при смене класса.

>>> a.\_\_dict\_\_

{'name': 'a'}

Доступ к классовым переменным и методам «прошлого» класса A пропал:

>>> a.foo()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'B' object has no attribute 'foo'

А вот классовые переменные и методы класса B доступы:

>>> a.bar()

bar

>>> a.qux

'B'

Работа с атрибутам объекта: установка, удаление и поиск, равносильна вызову встроенных функций settattr, delattr, getattr:

a.x = 1 <==> setattr(a, 'x', 1)

del a.x <==> delattr(a, 'x')

a.x <==> getattr(a, 'x')

При этом стоит стоит понимать, что setattr и delattr влияют и изменяют только сам объект (точнее a.\_\_dict\_\_), и не изменяют класс объекта.

qux — является классовой переменной, т.е. она «принадлежит» классу B, а не объекту a:

>>> a.qux

'B'

>>> a.\_\_dict\_\_

{'name': 'a'}

Если мы попытаемся удалить этот атрибут, то получим ошибку, т.к. delattr будет пытаться удалить атрибут из a.\_\_dict\_\_

>>> delattr(a, 'qux')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: qux

>>> del a.qux

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: qux

>>> a.qux

'B'

>>>

Далее, если мы попытаемся изменить (установить) атрибут, setattr поместит его в \_\_dict\_\_, специфичный для данного, конкретного объекта.

>>> b = B()

>>> b.qux

'B'

>>> a.qux = 'myB'

>>> a.qux

'myB'

>>> a.\_\_dict\_\_

{'qux': 'myB', 'name': 'a'}

>>> b.qux

'B'

>>>

Ну и раз есть 'qux' в \_\_dict\_\_ объекта, его можно удалить с помощью delattr:

>>> del a.qux

После удаления, a.qux будет возвращать значение классовой переменной:

>>> a.qux

'B'

>>> a.\_\_dict\_\_

{'name': 'a'}

Итак:

класс для объекта — это значение специального атрибута \_\_class\_\_ и его можно менять. (Хотя в официальной документации говорится, что никаких гарантий нет, но на самом деле можно)

почти каждый объект имеет свое пространство имен (атрибутов), доступ (не всегда полный), к которому осуществляется с помощью специального атрибута \_\_dict\_\_

класс фактичеки влияет только на поиск атрибутов, которых нет в \_\_dict\_\_, как-то: методы класса, дескрипторы, магические методы, классовые переменные и прочее.

Объекты и классы

Классы — это объекты, и у них тоже есть специальные атрибуты \_\_class\_\_ и \_\_dict\_\_.

>>> class A(object):

... pass

...

У класса тип type.

>>> A.\_\_class\_\_

<type 'type'>

Правда \_\_dict\_\_ у классов не совсем словарь

>>> A.\_\_dict\_\_

<dictproxy object at 0x1111e88>

Но \_\_dict\_\_ ответственен за доступ к внутреннему пространству имен, в котором хранятся методы, дескрипторы, переменные, свойства и прочее:

>>> dict(A.\_\_dict\_\_)

{'\_\_module\_\_': '\_\_main\_\_', 'qux': 'A', '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'A' objects>, 'foo': <function foo at 0x7f7797a25c08>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'A' objects>, '\_\_doc\_\_': None}

>>> A.\_\_dict\_\_.keys()

['\_\_module\_\_', 'qux', '\_\_dict\_\_', 'foo', '\_\_weakref\_\_', '\_\_doc\_\_']<

В классах помимо \_\_class\_\_ и \_\_dict\_\_, имеется еще несколько специальных атрибутов: \_\_bases\_\_ — список прямых родителей, \_\_name\_\_ — имя класса. [1]

Классы можно считать эдакими расширениями обычных объектов, которые реализуют интерфейс типа. Множество всех классов (или типов) принадлежат множеству всех объектов, а точнее является его подмножеством. Иначе говоря, любой класс является объектом, но не всякий объект является классом. Договоримся называть обычными объектами(regular objects) те объекты, которые классами не являются.

Небольшая демонстрация, которая станет лучше понятна чуть позже.

Класс является объектом.

>>> class A(object):

... pass

...

>>> isinstance(A, object)

True

Число — это тоже объект.

>>> isinstance(42, object)

True

Класс — это класс (т.е. тип).

>>> isinstance(A, type)

True

А вот число классом (типом) не является. (Что такое type будет пояснено позже)

>>> isinstance(42, type)

False

>>>

Ну и a — тоже обычный объект.

>>> a = A()

>>> isinstance(a, A)

True

>>> isinstance(a, object)

True

>>> isinstance(a, type)

False

И у A всего один прямой родительский класс — object.

>>> A.\_\_bases\_\_

(<type 'object'>,)

Часть специальных параметров можно даже менять:

>>> A.\_\_name\_\_

'A'

>>> A.\_\_name\_\_ = 'B'

>>> A

<class '\_\_main\_\_.B'>

С помощью getattr получаем доступ к атрибутам класса:

>>> A.qux

'A'

>>> A.foo

<unbound method A.foo>

>>>

Поиск атрибутов в обычном объекте

В первом приближении алгоритм поиска выглядит так: сначала ищется в \_\_dict\_\_ объекта, потом идет поиск по \_\_dict\_\_ словарям класса объекта (который определяется с помощью \_\_class\_\_) и \_\_dict\_\_ его базовых классов в рекурсивном порядке.

Пример.

>>> class A(object):

... qux = 'A'

... def \_\_init\_\_(self, name):

... self.name=name

... def foo(self):

... print 'foo'

...

>>> a = A()

>>> b = A()

Т.к. в обычных объектах a и b нет в \_\_dict\_\_ атрибута 'qux', то поиск продолжается во внутреннем словаре \_\_dict\_\_ их типа (класса), а потом по \_\_dict\_\_ словарям родителей в определенном порядке:

>>> b.qux

'A'

>>> A.qux

'A'

Меняем атрибут qux у класса A. И соответственно должны поменяться значения, которые возвращают экземпляры класса A — a и b:

>>> A.qux='B'

>>> a.qux

'B'

>>> b.qux

'B'

>>>

Точно так же в рантайме к классу можно добавить метод:

>>> A.quux = lambda self: 'i have quux method'

>>> A.\_\_dict\_\_['quux']

<function <lambda> at 0x7f7797a25b90>

>>> A.quux

<unbound method A.<lambda>>

И доступ к нему появится у экземпляров:

>>> a.quux()

'i have quux method'

Точно так же как и с любыми другими объектами, можно удалить атрибут класса, например, классовую переменную qux:

>>> del A.qux

Она удалиться из \_\_dict\_\_

>>> A.\_\_dict\_\_['qux']

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

KeyError: 'qux'

И доступ у экземляров пропадет.

>>> a.qux

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'A' object has no attribute 'qux'

>>>

У классов почти такой же поиск атрибутов, как и у обычных объектов, но есть отличия: поиск начинается с собственного \_\_dict\_\_ словаря, а потом идет поиск по \_\_dict\_\_ словарям суперклассов (которые хранятся в \_\_bases\_\_) по опредленному алгоритму, а затем по классу в \_\_class\_\_ и его суперклассах. (Подробнее об этом позже).

Cсылки

Unifying types and classes in Python — главный документ, объясняющий что, как и зачем в новых классах.

Making Types Look More Like Classes — PEP 252, описывающий отличие старых классов от новых.

Built-in functions — детальное описание работы всех встроенных функций.

Data model — детальное описание модели данных python'а.

Python types and objects — объяснение объектной модели python на простых примерах с картинками.

Примечания

[1] О \_\_module\_\_ и \_\_doc\_\_ для простоты изложения пока забудем. Полный список атрибутов класса можно посмотреть в документации