

part17

委譲 (デリゲート) での `__getattr__` の使い方が分りません。 (19-39)

19 名前: デフォルトの名無しさん 投稿日: 2007/02/18(日) 01:54:54
委譲 (デリゲート) での `__getattr__` の使い方が分りません。
従来の `__getattr__()` は、解決できない属性名の時だけ呼ばれるが
`__getattr__()` は、属性値の取得が必要になったときに、常に呼ばれるらしいのですが。

まず、`__getattr__` でのサンプル。

```
class wrapper(object):
    def __init__(self, obj):
        self.__wrapped = obj # obj を組み込む。
    def __getattr__(self, attrname): # attrname は、知らない名前属性が来たら呼び出される
        return getattr(self.__wrapped, attrname) # 組み込まれたオブジェクトに getattr 関数で処理を委託。
```

`l = wrapper([])` # リストを引数に wrapper クラスを呼び出すと、リストにできる操作なら全てできる

(実行すると、正しく動作します)

```
>>> l.append(333)
>>> l.pop(0)
333
```

20 名前: デフォルトの名無しさん 投稿日: 2007/02/18(日) 01:56:08
ここで、`__getattr__` を `__getattr__` に置き換えます。

```
class wrapper(object):
    def __init__(self, obj):
        self.__wrapped = obj # obj を組み込む。
    def __getattr__(self, attrname):
        return getattr(self.__wrapped, attrname)
```

`l = wrapper([])`

(試しに、実行してみます)

```
>>> l.append(333)
(ここで暴走)
```

なぜ暴走するのかよく分りません。
`__getattr__` はどう使うのが正しいのでしょうか?

21 名前: デフォルトの名無しさん 投稿日: 2007/02/18(日) 02:48:11
>>20
`self.__getattr__` の中で `self.__wrapped` を取得しようとする
さらに `self.__getattr__(self, '_wrapped')` を呼び出して無限再帰ループになるみたい。

解決方法は・・・たとえば、`__wrapped` をインスタンス属性としてもつんじゃなくて
クラス属性にインスタンスをキーにした辞書をもつようにするとかどう?

```
class wrapper(object):
    __wrapped = {}
    def __init__(self, obj):
        wrapper.__wrapped[hash(self)] = obj
    def __del__(self):
        del wrapper.__wrapped[hash(self)]
    def __getattr__(self, attrname):
        return getattr(wrapper.__wrapped[hash(self)], attrname)
```

23 名前: デフォルトの名無しさん 投稿日: 2007/02/18(日) 03:00:57
`__getattr__` を `__getattr__` に置き換えるって話じゃないのか

```
24 名前: デフォルトの名無しさん 投稿日: 2007/02/18(日) 03:08:57
if attrname not in ['_wrapped', etc, etc]:
    return getattr(self.__wrapped, attrname)
else:
    super(wrapper, self).__getattr__(attrname)
```

25 名前: デフォルトの名無しさん 投稿日: 2007/02/18(日) 03:14:55
>>21 でおkかと思ったけど、`len()` ができないなあ。

```
>>> l = wrapper([])
>>> len(l)
```

```
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    len(l)
TypeError: object of type 'wrapper' has no len()
```

TypeError なあたり、決まった型にしか呼べないのかも。
len(x) って内部で x.__len__ を返してるのかと思ってたけど、ちょっと違うのかな。

```
>>> l.append(3)
>>> l.__len__()
1
```

27 名前：デフォルトの名無しさん 投稿日：2007/02/18(日) 04:03:11

>>25
アクセスでフィールド値を取得してから、len しないとダメじゃない？
__getattr__ じゃなく、__getatrr__ の例で悪いけど、
この場合、同じと思う。

```
class wrapper(object):
    def __init__(self, obj):
        self.__wrapped = obj
    def __getatrr__(self, attrname):
        return getattr(self.__wrapped, attrname)
    def get_wrapped(self): # フィールド値取得メソッドを設定
        return self.__wrapped
```

```
l = wrapper([])
l.append(333)
```

(ここで len を実行。正常に動作する)

```
>>> len(l.get_wrapped())
1
>>>
```

28 名前：デフォルトの名無しさん 投稿日：2007/02/18(日) 04:37:24

>>25
__getattr__ でフックされるのは、dir(l) で出てくるものだけだと思う。
したがって、len は使えないが __len__ は使えるということになる。

```
>>> dir(l)
['_add_', '__class__', '__contains__', '__delattr__',
'_delitem_', '__delslice__', '__doc__', '__eq__', '__ge__',
'_getattr__', '__getitem__', '__getslice__', '__gt__',
'_hash_', '__iadd__', '__imul__', '__init__', '__iter__',
'_le_', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
'_reduce_', '__reduce_ex__', '__repr__', '__reversed__',
'_rmul_', '__setattr__', '__setitem__', '__setslice__',
'_str_', 'append', 'count', 'extend', 'index', 'insert',
'pop', 'remove', 'reverse', 'sort']
>>>
```

30 名前：デフォルトの名無しさん 投稿日：2007/02/18(日) 04:50:46

>>28
なるほど。__len__ が特別だったのか。確かに __init__ とかフックされたら困るよなあ。
__len__ を定義して、ラップしているオブジェクトの len() を返すようにしたら、ちゃんと動きました。
質問主じゃないけど勉強になった。この先 __getattr__ を使うかどうかはともかくとして w

35 名前：デフォルトの名無しさん 投稿日：2007/02/18(日) 12:43:37

>>19 >>20 で __getattr__ の使い方について質問した者です。

なぜ、__getatrr__ ではなく __getattr__ を使いたかったかということ、
「スーパークラスの配列で、複数の異なるサブクラスのインスタンスを一元管理する」
というテクニックを使えるようにしておきたいと思ったからです。

ちなみに、従来からあるメソッド __getatrr__() は、解決できない属性名の時だけ呼ばれるが
__getattr__() は、属性値の取得が必要になったときに、常に呼ばれる、とのこと。

「スーパークラスの配列で、複数の異なるサブクラスのインスタンスを一元管理する」に
ついては、java の解説ページですが、下のところにあります。
<http://itpro.nikkeibp.co.jp/members/NSW/ITBASIC/20050620/162997/>

(略) 部長さん、課長さん、担当さんをそれぞれ表す Bucho クラス、Kacho クラス、Tanto クラスを
定義し、それぞれのクラスのメンバーとして給与の金額を返す getKyuyo という関数があるとします。

複数のクラスに同じ名前の関数があるのですから、それらを汎化して社員を表す Shain クラスを定義しましょう。(略)

サブクラスである Bucho クラス, Kacho クラス, Tanto クラスは, 0 を返す getKyuyo を継承することになります。こんな関数を継承しても役に立ちませんね。スーパークラスから継承した関数の機能がサブクラスに合わない場合は, サブクラスで同名のメソッドを記述すれば上書き変更できます。これを「オーバーライド (override)」と呼びます。部長さんの給与は 70 万円, 課長さんの給与は 50 万円, 担当さんの給与は 30 万円としましょう

(略) 汎化を行わなかったら「役職ごとに給与を求めて集計する」という手順になるでしょう。汎化を行ったことで「社員の給与を一気に集計する」という手順が実現できるのです。(終)

まあ、それに必要なと思ったんですが、単に shain = [] に shain.append(shainObject) で集めて、for で 要素に次々と .getKyuyo して、それを加算していだけでいいような気がします。

本当に __getattr__ が使えないと困るのかな? 頭冷やして考えないと分からないw とにかく、いろいろ参考になりました。みなさん、どうもありがとう。

37 名前: デフォルトの名無しさん 投稿日: 2007/02/18(日) 14:06:03

>>35

そのプログラムは単にオブジェクト指向のポリモーフィズム(多様性)を利用しているだけなので __getattr__ は必要ありません。そのプログラムを Python で書くなら、こんな感じになります。

```
class Shain:
    def getKyuyo(self): return 0
class Bucho(Shain):
    def getKyuyo(self): return 700000
class Kacho(Shain):
    def getKyuyo(self): return 500000
class Tanto(Shain):
    def getKyuyo(self): return 300000

s = [ Bucho(), Kacho(), Kacho(), Tanto(), Tanto(), Tanto() ]

goukei = 0
for p in s: goukei += p.getKyuyo()
print goukei
```

ちなみに、Python では Bucho Kacho Tanto を Shain のサブクラスにしなくても動きます。これは Java の「変数の型」という概念が Python にはないからです。Python ではオブジェクトに対してメソッドを呼び出した場合、その名前を持つメソッドが動的に(実行時に)決定されるので、スーパークラスで抽象化する必要はありません。(ダックタイピング)(ただし、スーパークラスの振る舞いを引き継ぐ為の継承は有効な手段です)呼び出されるメソッドが静的に(コンパイル時に)決定される Java や C++ などとの大きな違いですね。

39 名前: 35 投稿日: 2007/02/18(日) 14:26:56

>>37

サンクス。

a=1.2+1.0000000000000000ってやると ... (875-879)

875 名前: デフォルトの名無しさん 投稿日: 2007/04/03(火) 01:34:43

```
a=1.2
a=1.2+1.0000000000000000
ってやると a が
2.2000000000000000
になって困ってます
どうやったら解決しますか？
```

876 名前: デフォルトの名無しさん 投稿日: 2007/04/03(火) 01:39:54

```
>>> from decimal import Decimal
>>> Decimal('1.2') + Decimal('1.0000000000000000')
Decimal("2.2000000000000000")
```

878 名前: デフォルトの名無しさん 投稿日: 2007/04/03(火) 02:17:09
ありがとうございます

879 名前: デフォルトの名無しさん 投稿日: 2007/04/03(火) 08:02:41
ほんとにばいそん信者は使えないねw

チミ達は IDE というかエディターなどというものはなにに使ってる (915-922)

915 名前：デフォルトの名無しさん 投稿日：2007/04/05(木) 18:35:18
チミ達は IDE というかエディターなどというものはなにに使ってる
まさかメモ帳はありえないだろ普通www
おれは scite を使ってます

916 名前：デフォルトの名無しさん 投稿日：2007/04/05(木) 19:05:00
vim-7

917 名前：デフォルトの名無しさん 投稿日：2007/04/05(木) 19:10:29
xyzyy

918 名前：デフォルトの名無しさん 投稿日：2007/04/05(木) 19:21:39
IDLE

919 名前：デフォルトの名無しさん 投稿日：2007/04/05(木) 20:45:08
emacs

five-things-i-hate ってちゃんと使ってないと怖くて書けないよな

920 名前：デフォルトの名無しさん 投稿日：2007/04/05(木) 21:06:43
PyScripter + IDLE

921 名前：デフォルトの名無しさん 投稿日：2007/04/05(木) 21:18:10
秀丸

922 名前：デフォルトの名無しさん 投稿日：2007/04/05(木) 23:21:04
秀丸 たまに PyCrust 併用
最終確認が必要な場合は eclipse など

int を float にするほうほうがわかりません (931-933)

931 名前：デフォルトの名無しさん 投稿日：2007/04/06(金) 21:57:30
int を float にするほうほうがわかりません

932 名前：デフォルトの名無しさん 投稿日：2007/04/06(金) 22:05:59
>>>float(4)
4.0

933 名前：デフォルトの名無しさん 投稿日：2007/04/06(金) 22:09:14
ありがとうございます
とてもさんこうになります
カッコのなかは文字じゃないとだめとおもってました