whacorator

@Walheimat

January 13, 2023

Contents

1 Etymology

A **whacorator**, pronounced [wak-uh-rey-ter], is a method decorator to provide a class with a shared arity cache.

A method invocation represents a "whack" to its signature.

2 Summary

Repeated invocations of one method decorated with a whacorator incrementally reduce its arity; invocations of a second decorated method increases it again for any other decorated method.

This way, reducing one method's signature (as a count of its parameters), re-increases another's. And an endless game of *whac-a-mole* begins.

3 Specification

Note that **Python** syntax was chosen because of the popularity of decorators in the language.

Not every language has decorator support; in such languages other features (like AST-manipulating meta-programming) could be used to achieve the same effect.

3.1 Class Decorator vs. Method Decorator

A whacorator is a set of at least *three* decorators:

1. A class decorator to control shared caching
2. A first method decorator
3. A second method decorator

The size of the cache should be a non-zero integer.

```
@whacoratorclass
class UselessClass:

    @whacorator
    def thing_and_count(self, thing: str, count: int):
        return f"I have {count} {thing}(s)"

    def addition(self, a: int, b: int):
        return a + b

    @whacorator
    def say_name(self, name: str):
        return f"My name is {name}"

    @whacorator
    def favorite_number(self, number: int):
        return f"My favorite number is {number}"
```

In the example above, methods `thing_and_count`, `say_name` and `favorite_number` would share a whacorator cache.

The cache controls the behavior of arity loss and gain described below.

3.2 Arity Loss and Gain on Invocation

Using the example above, a sequence of invocations would yield the results below:

```
u = UselessClass()

u.thing_and_count("apple", 2)      # I have 2 apple(s), 2 is now cached
u.thing_and_count("orange", 3)     # I have 2 orange(s), orange is now cached

u.say_name("Krister")              # My name is Krister, thing_and_count regains 1

u.thing_and_count("lion", 200)     # I have 2 lion(s)
u.addition(1, 2)
u.addition(4, 3)
u.thing_and_count("confusion", 20) # I have 2 lion(s)

u.say_name("Ralph")                # My name is Ralph
u.say_name("Laura")                # My name is Ralph
u.say_name("Cem")                  # My name is Ralph
u.thing_and_count("clarity" 7)     # I have 2 clarity(s)

u.favorite_number(9)               # My favorite number is 12
u.thing_and_count("time", 4)       # I have 4 time(s)
u.favorite_number(13)              # My favorite number is 13
```

Repeated invocations of a method starts defaulting parameters *from right to left* ignoring passed arguments. Once lost, re-gaining arity is only possible if **another** decorated method loses arity.

In the example above, if `thing_and_count` has lost two parameters, repeated calls to `say_name` can only reinstate a single parameter since `say_name` itself can only lose one parameter.

Invocation of udndecorated methods doesn't influence the state of the cache.

3.2.1 Class Instance References

If class methods have a reference to the instance as a parameter, the whacorator does not include it in the cache. In the Python example, no method can lose its `self` parameter this way. So if `self.data` changes, it will also change if used in a method that has been invoked `parameter count + n` times.

3.3 State Sharing Between Instances

The whacorator cache is that of the *class* not its instances. That means if a method's arity was reduced by invocating it on one instance, another newly created instance will be affected y the same arity reduction.

```
a = Useless()

a.say_name("Krister")

b = Useless()

b.say_name("Sabine")            # My name is Krister
b.thing_and_count("tiger", 4)   # I have 4 tiger(s)
a.thing_and_count("lion", 2)    # I have 4 lion(s)
b.thing_and_count("giraffe", 9) # I have 4 lion(s)
b.say_name("Thomas")            # My name is Thomas
a.say_name("Cem")               # My name is Thomas
a.thing_and_count("gun", 12)    # I have 4 gun(s)
```

3.4 Squeaky Hammer

The class decorator should implicitly create class properties to inspect the current state of caching.

```
Useless.whacorator.say_name.arity # returns 1

u = Useless()
u.say_name("Krister")

Useless.whacorator.say_name.arity # returns 0
Useless.whacorator.say_name.cache # returns ["Krister"]
```