Praktische Anwendung von Metaklassen

Mirko Dziadzka

http://mirko.dziadzka.de/
@MirkoDziadzka

PyCon DE 2012

Wer bin ich?

- Bewege mich mit Softwareentwicklung auf dem Gebiet Unix, Netzwerk, Security
- Benutze Python seit ca. 1995
- ... in Projekten mit $> 100\,000$ LOC
- Zur Zeit bei Riverbed Technology in Regensburg

Wer seid ihr?

- Wer weiss, was Metaklassen sind?
- Wer schon mal welche benutzt?
- Wer benutzt Dekoratoren?
- Wer benutzt Klassen Dekoratoren?

Was sind Metaklassen?

Was sind Metaklassen?

Metaclasses are deeper magic than 99% of users should ever worry about. If you wonder whether you need them, you don't (the people who actually need them know with certainty that they need them, and don't need an explanation about why).
– Python Guru Tim Peters

Was sind Metaklassen?

- Ich glaube, es ist nicht ganz so schlimm ...
- 99% Prozent brauchen keine Metaklassen, sie können aber helfen, Probleme auf einfachere Art zu lösen

Was sind Metaklassen

Also ... was sind Metaklassen?

- Klassen in Python sind auch nur Objekte
- also Instanz einer Klasse
- Die Klasse von Klassen ist ‘type’

Was sind Metaklassen

```
>>> class Foo(object):
...     pass
>>> foo = Foo()
>>> foo
<__main__.Foo object at 0x1005b5b90>
>>> foo.__class__
<class '__main__.Foo'>
>>> foo.__class__.__class__
<type 'type'>
```

Was sind Metaklassen

- Die Klasse einer Benutzerdefinierten Klasse ist also *type*
- Wenn eine Klasse auch nur die Instanz einer anderen (Meta-)Klasse ist, so kann ich auch die Metaklasse benutzen, um eine Klasse zu erzeugen.

```
# define a class the 'normal' way
class Foo:
    default = 42
    def get_answer(self):
        return self.default

# define a class as object from type 'type'
def _Bar_get_answer(self):
    return self.default

Bar = type('Bar', (), {
    'default' : 42,
    'get_answer' : _Bar_get_answer,
})
```

Was sind Metaklassen

- uns das ist auch genau das, was intern passiert
- Python benutzt *type* um die Klasse zu erzeugen.
- Das lässt sich über das Attribute *__metaclass__* in der Klasse oder im Modul ändern
- Es reicht, wenn __metaclass__ in einer Basis Klasse deklariert wird
  - Hinweis: In grossen Projekten immer alles von einer (auch gerne leeren) Basisklasse ableiten

```
class Foo(object):
    answer = 42

# ist gleichwertig zu

class Foo(object):
    __metaclass__ = type
    answer = 42

# ist (fast) gleichwertig zu

class MetaClass(type):
    pass
class Foo(object):
    __metaclass__ = MetaClass
    answer = 42
```

Wie implementiert man eine Metaklasse

- Wie jede andere Klasse auch
- üblichwerweise definiert man die *__init__* oder die *__new__* Methode um seinen code unterzubringen.
- *__new__* scheint mir angebrachter, da es der frühest mögliche Zeitpunkt ist

```
class ExampleMetaClass(type):
    def __new__(cls, name, bases, attributes):
        res =  type(name, bases, attributes)
        print 'class %s created: %s' % (name, res)
        return res

class Foo(object):
    __metaclass__ = ExampleMetaClass

    def __init__(self):
        return
```

```
class Foo created: <class '__main__.Foo'>
```

Wie implementiert man eine Metaklasse

- In dieser *__new__* Methode kann man nun anfangen, die Klasse zu manipulieren
- ... oder zu analysieren

Beispiele

Wo benutzt man Metaklassen

Wo benutzt man Metaklassen?

- Typischerweise bei Implemenationen von Interfaces
  - ORM Mapper (django)
- Sprachmodifikationen (schlechter Ruf?)
- Debugging, ...

Debugging mit Metaklassen: trace

Das Problem

- Software wird an Kunden ausgeliefert.
- Falls es Probleme geben sollte, wünscht sich der Entwickler ausgiebige trace Informationen.

Standard Lösung

- *logging.debug* calls, überall wo es sinnvoll sein könnte

Grenzen der Lösung

- Debug Aufrufe "kosten" Rechenzeit
- Es ist nicht immer vorher klar, was sinnvoll ist
- Der Code wird durch debug Informationen unterbrochen und ist dadurch nicht so flüssig zu lesen

Debugging mit Metaklassen: trace

Wunschlösung

- bei Nichtbenutzung, keine Laufzeiteinflüsse
- selektives tracen von allen Methoden aufrufen und Ergebnissen im Programm
- Kunde kann Problem mit eingeschaltetem Debug reproduzieren und aussagekräftige Logs an die Entwickeler schicken

Debugging mit Metaklassen: trace

Idee:

- mittels config file / environment variable können beim Programmstart *interessante* Klassen/Methoden-namen angegeben werden
- eine Metaklasse dekoriert alle diese Methoden mit einem *trace* Dekorator

Debugging mit Metaklassen: trace

```
class BaseMetaClass(type):
    def __init__(self, clsname, bases, attr):
        for name, value in attr.items():
            if name.startswith('__'):
                continue
            fullname = "%s.%s.%s"  % (
                       self.__module__, clsname, name)
            if not should_wrap(fullname):
                continue
            value = trace_decorator(fullname, value)
            setattr(self, name, value)

class BaseClass(object):
    __metaclass__ = BaseMetaClass
```

Debugging mit Metaklassen: trace

```
import os
import fnmatch

DEBUG_TARGETS = os.environ.get('DEBUG_TARGETS','')

def should_wrap(fullname):
    for pattern in DEBUG_TARGETS.split(','):
        if fnmatch.fnmatch(fullname, pattern):
            return True
    return False
```

Debugging mit Metaklassen: trace

```
def log(message, *args):
    print message % args

def trace_decorator(fullname, method):
    def wrapper(*args, **kwargs):
        log("%s CALL %s %s", fullname, args, kwargs)
        try:
            res =  method(*args, **kwargs)
            log("%s RETURN %s", fullname, res)
            return res
        except Exception,e:
            log("%s EXCEPTION %s", fullname, str(e))
            raise
    #...
    return wrapper
```

Debugging mit Metaklassen: trace

Anwendung

```
$ DEBUG_TARGETS=
$ ./metaclass_debug_usage.py
$ DEBUG_TARGETS=*.Question.*
$ ./metaclass_debug_usage.py
__main__.Question.set_answer CALL (<__main__.Question object at 0x102cb1ad0>, 2
__main__.Question.set_answer RETURN None
```

Andere Anwendungen von Metaklassen

- Dynamische Code Analyse
  - Callgraph
  - Von wo wird auf eine Variable lesend/schreibend zugegriffen
  - Coding Style enforcement
- fault injection

Weiterführendes

Hintergrundinformationen

http://stackoverflow.com/questions/100003/what-is-a-metaclass-in-python
http://www.ibm.com/developerworks/linux/library/l-pymeta/index.html
http://cleverdevil.org/computing/78/

Beispielcode

https://github.com/MirkoDziadzka/pycon-meta

Folien als PDF

http://mirko.dziadzka.de/Vortrag/pycon-meta-20121030.pdf

Danke für's zuhören

Fragen?