dataclass_modelclass

June 19, 2024

[1]:
```
from dataclasses import dataclass
from typing import Optional


@dataclass
class Foo:
    a: Optional[int] = None
    b: Optional[str] = None
    @staticmethod
    def from_dict(d: dict):
        return Foo(
            d.get("a", None),
            d.get("b", None),
        )
```

[2]:
```
f = Foo(4, 3)
f.a, f.b
```

[2]:
```
(4, 3)
```

[3]:
```
f = Foo(2)
f.a, f.b
```

[3]:
```
(2, None)
```

[4]:
```
f = Foo(a=3, b=4)
f.a, f.b
```

[4]:
```
(3, 4)
```

[5]:
```
f = Foo(a=4, 2)
```

```
  Cell In[5], line 1
    f = Foo(a=4, 2)
                 ^
SyntaxError: positional argument follows keyword argument
```

[6]:
```
f.__dict__
```

[6]:
```
{'a': 3, 'b': 4}
```

[7]:
```
f = Foo.from_dict({"a": 4})
f.a, f.b
```

[7]:
```
(4, None)
```

[8]:
```
import inspect
import typing
from dataclasses import dataclass


_T = typing.TypeVar("_T")


def modelclass(cls: typing.Type[_T]) -> typing.Type[_T]:
    cls = dataclass(cls)
    attributes = [
        a
        for a in cls.__dict__["__annotations__"].keys()
        if not a.startswith("__") and not inspect.isroutine(a)
    ]

    def init(self, *args, **kwargs):
        for i, a in enumerate(args):
            if i < len(attributes):
                self.__dict__[attributes[i]] = a
        for k, v in kwargs.items():
            if k in attributes:
                self.__dict__[k] = v

    cls.__init__ = init  # type: ignore[assignment]

    return cls
```

[9]:
```
@modelclass
class Foo:
    a: Optional[int] = None
    b: Optional[str] = None
    @staticmethod
    def from_dict(d: dict):
        return Foo(
            d.get("a", None),
            d.get("b", None),
        )
```

[10]:
```
f = Foo(4, 3)
f.a, f.b
```

[10]:
```
(4, 3)
```

[11]:
```
f = Foo(2)
f.a, f.b
```

[11]:
```
(2, None)
```

[12]:
```
f = Foo(a=3, b=4)
f.a, f.b
```

[12]:
```
(3, 4)
```

[13]:
```
f = Foo(a=4, 2)
```

```
  Cell In[13], line 1
    f = Foo(a=4, 2)
                   ^
SyntaxError: positional argument follows keyword argument
```

[14]:
```
f.__dict__
```

[14]:
```
{'a': 3, 'b': 4}
```

[15]:
```
f = Foo.from_dict({"a": 4})
f.a, f.b
```

[15]:
```
(4, None)
```