# 9.13 使用元类控制实例的创建¶

## 问题¶

你想通过改变实例创建方式来实现单例、缓存或其他类似的特性。

## 解决方案¶

Python程序员都知道，如果你定义了一个类，就能像函数一样的调用它来创建实例，例如：

```
class Spam:
    def __init__(self, name):
        self.name = name

a = Spam('Guido')
b = Spam('Diana')
```

如果你想自定义这个步骤，你可以定义一个元类并自己实现 `__call__()` 方法。

为了演示，假设你不想任何人创建这个类的实例：

```
class NoInstances(type):
    def __call__(self, *args, **kwargs):
        raise TypeError("Can't instantiate directly")

# Example
class Spam(metaclass=NoInstances):
    @staticmethod
    def grok(x):
        print('Spam.grok')
```

这样的话，用户只能调用这个类的静态方法，而不能使用通常的方法来创建它的实例。例如：

```
>>> Spam.grok(42)
Spam.grok
>>> s = Spam()
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "example1.py", line 7, in __call__
        raise TypeError("Can't instantiate directly")
TypeError: Can't instantiate directly
>>>
```

现在，假如你想实现单例模式（只能创建唯一实例的类），实现起来也很简单：

```
class Singleton(type):
    def __init__(self, *args, **kwargs):
        self.__instance = None
        super().__init__(*args, **kwargs)

    def __call__(self, *args, **kwargs):
        if self.__instance is None:
            self.__instance = super().__call__(*args, **kwargs)
            return self.__instance
        else:
            return self.__instance

# Example
class Spam(metaclass=Singleton):
    def __init__(self):
        print('Creating Spam')
```

那么Spam类就只能创建唯一的实例了，演示如下：

```
>>> a = Spam()
```

```
Creating Spam
>>> b = Spam()
>>> a is b
True
>>> c = Spam()
>>> a is c
True
>>>
```

最后，假设你想创建8.25小节中那样的缓存实例。下面我们可以通过元类来实现：

```
import weakref

class Cached(type):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.__cache = weakref.WeakValueDictionary()

    def __call__(self, *args):
        if args in self.__cache:
            return self.__cache[args]
        else:
            obj = super().__call__(*args)
            self.__cache[args] = obj
            return obj

# Example
class Spam(metaclass=Cached):
    def __init__(self, name):
        print('Creating Spam({!r})'.format(name))
        self.name = name
```

然后我也来测试一下：

```
>>> a = Spam('Guido')
Creating Spam('Guido')
>>> b = Spam('Diana')
Creating Spam('Diana')
>>> c = Spam('Guido') # Cached
>>> a is b
False
>>> a is c # Cached value returned
True
>>>
```

# 讨论¶

利用元类实现多种实例创建模式通常要比不使用元类的方式优雅得多。

假设你不使用元类，你可能需要将类隐藏在某些工厂函数后面。 比如为了实现一个单例，你你可能会像下面这样写：

```
class _Spam:
    def __init__(self):
        print('Creating Spam')

_spam_instance = None

def Spam():
    global _spam_instance

    if _spam_instance is not None:
        return _spam_instance
    else:
        _spam_instance = _Spam()
        return _spam_instance
```

尽管使用元类可能会涉及到比较高级点的技术，但是它的代码看起来会更加简洁舒服，而且也更加直观。

更多关于创建缓存实例、弱引用等内容，请参考8.25小节。