9.15 定义有可选参数的元类¶

问题¶

你想定义一个元类，允许类定义时提供可选参数，这样可以控制或配置类型的创建过程。

解决方案¶

在定义类的时候，Python允许我们使用 ``metaclass``关键字参数来指定特定的元类。 例如使用抽象基类：

```
from abc import ABCMeta, abstractmethod
class IStream(metaclass=ABCMeta):
    @abstractmethod
    def read(self, maxsize=None):
        pass

    @abstractmethod
    def write(self, data):
        pass
```

然而，在自定义元类中我们还可以提供其他的关键字参数，如下所示：

```
class Spam(metaclass=MyMeta, debug=True, synchronize=True):
    pass
```

为了使元类支持这些关键字参数，你必须确保在 `__prepare__()`, `__new__()` 和 `__init__()` 方法中 都使用强制关键字参数。就像下面这样：

```
class MyMeta(type):
    # Optional
    @classmethod
    def __prepare__(cls, name, bases, *, debug=False, synchronize=False):
        # Custom processing
        pass
        return super().__prepare__(name, bases)

    # Required
    def __new__(cls, name, bases, ns, *, debug=False, synchronize=False):
        # Custom processing
        pass
        return super().__new__(cls, name, bases, ns)

    # Required
    def __init__(self, name, bases, ns, *, debug=False, synchronize=False):
        # Custom processing
        pass
        super().__init__(name, bases, ns)
```

讨论¶

给一个元类添加可选关键字参数需要你完全弄懂类创建的所有步骤， 因为这些参数会被传递给每一个相关的方法。 `__prepare__()` 方法在所有类定义开始执行前首先被调用，用来创建类命名空间。 通常来讲，这个方法只是简单的返回一个字典或其他映射对象。 `__new__()` 方法被用来实例化最终的类对象。它在类的主体被执行完后开始执行。 `__init__()` 方法最后被调用，用来执行其他的一些初始化工作。

当我们构造元类的时候，通常只需要定义一个 `__new__()` 或 `__init__()` 方法，但不是两个都定义。 但是，如果需要接受其他的关键字参数的话，这两个方法就要同时提供，并且都要提供对应的参数签名。 默认的 `__prepare__()` 方法接受任意的关键字参数，但是会忽略它们， 所以只有当这些额外的参数可能会影响到类命名空间的创建时你才需要去定义 `__prepare__()` 方法。

通过使用强制关键字参数，在类的创建过程中我们必须通过关键字来指定这些参数。

使用关键字参数配置一个元类还可以视作对类变量的一种替代方式。例如：

```
class Spam(metaclass=MyMeta):
    debug = True
    synchronize = True
    pass
```

将这些属性定义为参数的好处在于它们不会污染类的名称空间， 这些属性仅仅只从属于类的创建阶段，而不是类中的语句执行阶段。 另外，它们在 `__prepare__()` 方法中是可以被访问的，因为这个方法会在所有类主体执行前被执行。但是类变量只能在元类的 `__new__()` 和 `__init__()` 方法中可见。