8.12 定义接口或者抽象基类¶

问题¶

你想定义一个接口或抽象类，并且通过执行类型检查来确保子类实现了某些特定的方法

解决方案¶

使用 `abc` 模块可以很轻松的定义抽象基类：

```
from abc import ABCMeta, abstractmethod

class IStream(metaclass=ABCMeta):
    @abstractmethod
    def read(self, maxbytes=-1):
        pass

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

抽象类的一个特点是它不能直接被实例化，比如你想像下面这样做是不行的：

```
a = IStream() # TypeError: Can't instantiate abstract class
                # IStream with abstract methods read, write
```

抽象类的目的就是让别的类继承它并实现特定的抽象方法：

```
class SocketStream(IStream):
    def read(self, maxbytes=-1):
        pass

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

抽象基类的一个主要用途是在代码中检查某些类是否为特定类型，实现了特定接口：

```
def serialize(obj, stream):
    if not isinstance(stream, IStream):
        raise TypeError('Expected an IStream')
    pass
```

除了继承这种方式外，还可以通过注册方式来让某个类实现抽象基类：

```
import io

# Register the built-in I/O classes as supporting our interface
IStream.register(io.IOBase)

# Open a normal file and type check
f = open('foo.txt')
isinstance(f, IStream) # Returns True
```

`@abstractmethod` 还能注解静态方法、类方法和 `properties` 。 你只需保证这个注解紧靠在函数定义前即可：

```
class A(metaclass=ABCMeta):
    @property
    @abstractmethod
    def name(self):
        pass

    @name.setter
    @abstractmethod
    def name(self, value):
        pass
```

```
    @classmethod
    @abstractmethod
    def method1(cls):
        pass

    @staticmethod
    @abstractmethod
    def method2():
        pass
```

讨论¶

标准库中有很多用到抽象基类的地方。`collections` 模块定义了很多跟容器和迭代器(序列、映射、集合等)有关的抽象基类。 `numbers` 库定义了跟数字对象(整数、浮点数、有理数等)有关的基类。`io` 库定义了很多跟I/O操作相关的基类。

你可以使用预定义的抽象类来执行更通用的类型检查，例如：

```
import collections

# Check if x is a sequence
if isinstance(x, collections.Sequence):
...

# Check if x is iterable
if isinstance(x, collections.Iterable):
...

# Check if x has a size
if isinstance(x, collections.Sized):
...

# Check if x is a mapping
if isinstance(x, collections.Mapping):
```

尽管ABCs可以让我们很方便的做类型检查，但是我们在代码中最好不要过多的使用它。 因为Python的本质是一门动态编程语言，其目的就是给你更多灵活性， 强制类型检查或让你代码变得更复杂，这样做无异于舍本求末。