4.2 代理迭代¶

问题¶

你构建了一个自定义容器对象，里面包含有列表、元组或其他可迭代对象。 你想直接在你的这个新容器对象上执行迭代操作。

解决方案¶

实际上你只需要定义一个 `__iter__()` 方法，将迭代操作代理到容器内部的对象上去。比如：

```
class Node:
    def __init__(self, value):
        self._value = value
        self._children = []

    def __repr__(self):
        return 'Node({!r})'.format(self._value)

    def add_child(self, node):
        self._children.append(node)

    def __iter__(self):
        return iter(self._children)

# Example
if __name__ == '__main__':
    root = Node(0)
    child1 = Node(1)
    child2 = Node(2)
    root.add_child(child1)
    root.add_child(child2)
    # Outputs Node(1), Node(2)
    for ch in root:
        print(ch)
```

在上面代码中， `__iter__()` 方法只是简单的将迭代请求传递给内部的 `_children` 属性。

讨论¶

Python的迭代器协议需要 `__iter__()` 方法返回一个实现了 `__next__()` 方法的迭代器对象。 如果你只是迭代遍历其他容器的内容，你无须担心底层是怎样实现的。你所要做的只是传递迭代请求既可。

这里的 `iter()` 函数的使用简化了代码， `iter(s)` 只是简单的通过调用 `s.__iter__()` 方法来返回对应的迭代器对象，就跟 `len(s)` 会调用 `s.__len__()` 原理是一样的。