`Serquential`, `ModuleList`,和 `ModuleDict`-3

`ModuleDict`

构造方法

`ModuleDict`的构造方法和`ModuleList`一样直接，逻辑也类似，唯一区别是调用`self.update()`方法，这也符合字典特性：

```
def __init__(self, modules: Optional[Mapping[str, Module]] = None) -> None:
  super(ModuleDict, self).__init__()
  if modules is not None:
    self.update(modules)
```

`update`方法也不复杂，省略掉一些类型判断和异常处理后，逻辑如下：

```
def update(self, modules: Mapping[str, Module]) -> None:
  if isinstance(modules, (OrderedDict, ModuleDict, container_abcs.Mapping)):
      for key, module in modules.items():
        self[key] = module
      else:
        for j, m in enumerate(modules):
              self[m[0]] = m[1]
```

> 这里使用`enumerate`也仅仅是因为`j`是在异常处理中的提示信息（这里没列出来）

其他魔术方法

`ModuleDict`的其他魔术方法可以说是最简单直接的，因为其操作对象`self._modules`的类型为`OrderDict`，因此很多魔术方法都十分直接的字典操作：

```
def __getitem__(self, key: str) -> Module:
  return self._modules[key]

def __setitem__(self, key: str, module: Module) -> None:
  self.add_module(key, module)

def __delitem__(self, key: str) -> None:
  del self._modules[key]

def __len__(self) -> int:
   return len(self._modules)

def __iter__(self) -> Iterator[str]:
   return iter(self._modules)

def __contains__(self, key: str) -> bool:
  return key in self._modules
```

字典特性方法

正如前文提到，因为其操作对象`self._modules`的类型为`OrderDict`，因此`ModuleDict`的一些字典特性方法也是直接对`_modules`的操作:

```
def clear(self) -> None:
  self._modules.clear()

def pop(self, key: str) -> Module:
  v = self[key]
  del self[key]
  return v

def keys(self) -> Iterable[str]:
  return self._modules.keys()

def items(self) -> Iterable[Tuple[str, Module]]:
  return self._modules.items()

def values(self) -> Iterable[Module]:
  return self._modules.values()
```

##

`ModuleDict`确实没有多少复杂的逻辑可写的..