from torch.utils.data import Dataset

# 假设 Dataset 类中有一个通用的初始化方法

class Dataset:

def \_\_init\_\_(self):

self通用属性 = "这是Dataset类的一个属性"

def 通用方法(self):

return "这是Dataset类的一个方法"

# COVID19Dataset 类继承自 Dataset

class COVID19Dataset(Dataset):

def \_\_init\_\_(self, data):

super().\_\_init\_\_() # 调用父类的初始化方法

# 其他 COVID19Dataset 特有的初始化代码

self.data = data

# 定义 COVID19Dataset 特有的方法

def 特定方法(self):

return "这是COVID19Dataset类的一个特有方法"

# 创建 COVID19Dataset 的实例

covid19\_dataset = COVID19Dataset(data="COVID-19 数据")

# 直接使用继承自 Dataset 的属性和方法

print(covid19\_dataset.通用属性) # 输出: 这是Dataset类的一个属性

print(covid19\_dataset.通用方法()) # 输出: 这是Dataset类的一个方法

# 使用 COVID19Dataset 特有的方法

print(covid19\_dataset.特定方法()) # 输出: 这是COVID19Dataset类的一个特有方法

在这个例子中：

我们假设 Dataset 类有一个名为 通用属性 的属性和一个名为 通用方法 的方法。

COVID19Dataset 类继承自 Dataset 类，因此在其构造器中通过 super().\_\_init\_\_() 调用了父类的构造器，从而初始化了继承来的属性。

即使没有显式地在 COVID19Dataset 中定义 通用方法，由于继承，COVID19Dataset 的实例 covid19\_dataset 仍然可以调用这个方法。

同时，COVID19Dataset 类也可以有自己的特有属性和方法，比如这里的 特定方法。

通过继承，COVID19Dataset 不仅能够使用父类 Dataset 提供的功能，还可以添加自己的特定功能，实现代码的复用和扩展。在实际使用中，PyTorch 的 Dataset 类提供了 \_\_len\_\_ 和 \_\_getitem\_\_ 等抽象方法，你需要在自定义数据集类中实现这些方法来指定如何获取数据集的大小以及如何按索引访问数据集中的项。