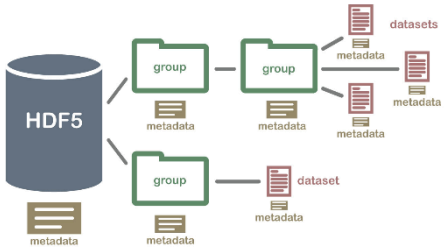
第五章 HDF5 与 NetCDF 文件

5.1 HDF5 与 NetCDF 数据格式



5.1.1 HDF5 数据格式

5.1.1.1 定义与背景

全称 **Hierarchical Data Format version 5 分层数据格式版本 5**

开发背景 由 HDF Group 开发，旨在存储和管理复杂、大规模数据，广泛应用于科学计算和工程仿真中。

适用模式 使用该文件格式的数值模式有：**MPAS、WRF、FV3** 等

依赖库 **zlib、hdf5、netcdf**

5.1.1.2 主要特点

特点

① **分层结构** 采用类似文件系统的层次组织形式，使用**组 group** 和数据集 dataset 构建数据模型。
支持**任意深度和复杂的嵌套结构**，每个组可以包含子组、数据集、元数据。
元数据：用来描述数据的数据（具备自我描述功能），即**局地属性 local attribute**
文件属性：**global attribute**，对整个文件进行描述，例如创建日期、内容、来源等

② **高效存储** 支持数据压缩、分块存储 chunking、部分加载
zlib 压缩工具，一般在写入时同时压缩。

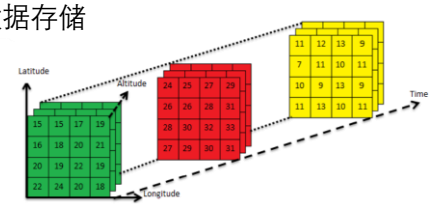
③ **灵活性** 存储多种数据类型（数值、字符、复合类型等），允许自定义数据结构。

④ **并行 I/O** 提供**并行读写能力**，适合高性能计算环境。
MPI-IO 库提供标准的并行读写

⑤ **跨平台** 数据文件在不同操作系统无缝传输使用

应用场景 科学研究、工程仿真、有限元分析、大规模数据处理、高性能计算数据存储

5.1.2 NetCDF 数据格式



5.1.2.1 定义与背景

全称 **Network Common Data Form** 网络通用数据表

开发背景 由 Unidata 开发，**专门用于气象、海洋、气候等科学数据**的存储与交换，设计目标是提供自描述且标准化的数据格式。

版本演变 NetCDF-3 **经典默认版本**，适用于基本的多维数组存储
NetCDF-4 引入 **HDF5 作为底层存储格式**，更容易安装

5.1.2.2 数据模型与结构

维度 定义数据数组的方向（如时间、经度、纬度），支持无限维度（例如时间序列数据）

变量 多维数组数据的存储单元，每个变量对应一个或多个纬度

属性 为文件、变量或维度提供描述信息，确保数据的自描述性

标准化 利用 CF (Climate and Forecast) 元数据约定，确保数据在不同平台的一致性和互操作性。

5.1.3 NetCDF-3/4 版本区别

5.1.3.1 数据模型与功能区别

NetCDF-3 基于维度 Dimensions、变量 Variables 和全局属性 Global Attributes，不支持组 Group 结构，所有变量存储在文件根目录下。适用于相对简单的数据存储需求。

NetCDF-4　兼容经典 NetCDF-3 模型，同时引入组 Group 和嵌套组的概念，可实现分层结构存储。支持复杂数据类型、压缩、分块存储以及并行 I/O。

5.1.3.2 依赖关系

NetCDF-4　① **基于 HDF5 格式**，要求系统中安装 **HDF5** 库。
② 如果需要并行 I/O，还可能需要 **MPI** 库支持。
③ Python 中常用的 **netCDF4** 模块是对 **NetCDF-4 C** 库的封装，既支持读取 NetCDF-3 文件，也支持 NetCDF-4 文件。

安装命令　`conda install -c anaconda netcdf4`

5.1.4 HDF5 与 NetCDF 的比较

共同点　① **多维数组存储**：均能高效存储多维数组数据，适合科学和工程数据。
② **自描述性**：都包含丰富的元数据（attributes），便于数据解释与共享。
③ **跨平台支持**：数据文件可以在不同操作系统间无缝传输。
④ **压缩与并行 I/O**：支持数据压缩、分块存储及一定程度的并行 I/O。

不同点　① **HDF5 通用性强**，适用于各种复杂数据结构；用户可自定义数据模型。NetCDF 专注于气象、海洋、气候等领域，数据模型标准化，强调数据的互操作性。
② **HDF5 采用层次结构**（组与数据集），灵活且可扩展。NetCDF 基于维度、变量和属性，结构固定

5.2 NetCDF 数据的读写

5.2.1 数据写入

气象数据特点　**多维性**　数据包含时间、纬度、经度三个维度
大数据量　数据来源可能为卫星、雷达、地面站、模式等
丰富的元数据　包括数据单位、观测时间、地理坐标、数据来源、质量控制信息等

基本流程　**定义数据维度→生成经纬度数据→创建文件→定义维度→创建变量→属性赋值→写入数据**

操作案例　`def write_meteorological_data(file_path,lat,lon,time,data):`（下方代码按流程顺序并行展示）

① 定义数据维度

```
time_points=10
lat_points=50
lon_points=100
```

```
#生成模拟数据
temperature=np.random.uniform(low=-20,high=50,size=(time_points,lat_points,lon_points))
humidity=np.random.uniform(low=0,high=100,size=(time_points,lat_points,lon_points))
pressure=np.random.uniform(low=950,high=1100,size=(time_points,lat_points,lon_points))
```

② 生成经纬度数据

```
latitude=np.linspace(-90,90,lat_points)
#如果用 numpy 数组，可以省略很多语法
longitude=np.linspace(-180,180,lon_points)
```

③ 创建文件

```
ds=nc.Dataset(file_path,'w',format='NETCDF4')
```

④ 定义维度

```
ds.createDimension('time',time_points)
ds.createDimension('lat',lat_points)
ds.createDimension('lon',lon_points)
```

⑤ 创建变量　f8:表示 8 个字节，64 位浮点型数据

```
times = ds.createVariable('time','f8',('time',))
lats = ds.createVariable('lat','f4',('lat',))
lons = ds.createVariable('lon','f4',('lon',))
temp = ds.createVariable('temperature','f4',('time','lat','lon',),zlib=True,complevel=9)
hum = ds.createVariable('humidity','f4',('time','lat','lon',),zlib=True,complevel=9)
prss = ds.createVariable('pressure','f4',('time','lat','lon',),zlib=True,complevel=9)
```

⑥ 全局属性

```
ds.title = 'Meteorological Data'      ds.institution = 'NUIST'      ds.source = '模拟数据'
ds.time_coverage_start = datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ')
ds.time_coverage_end = datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ')
```

⑥ 设置变量属性 元数据

```
times.units = 'seconds since 2025-01-01 00:00:00'
lats.units = 'degrees_north'
lons.units = 'degrees_east'
temp.units = '℃'
temp.long_name = 'Temperature'
hum.units = '%'
hum.long_name = 'Humidity'
prss.units = 'hPa'
prss.long_name = 'Pressure'
```

⑦ 写入数据

```
times[:] = np.arange(time_points)
lats[:] = latitude
lons[:] = longitude
temp[:] = temperature
hum[:] = humidity
prss[:] = pressure

ds.close()
```

5.2.2 数据读取

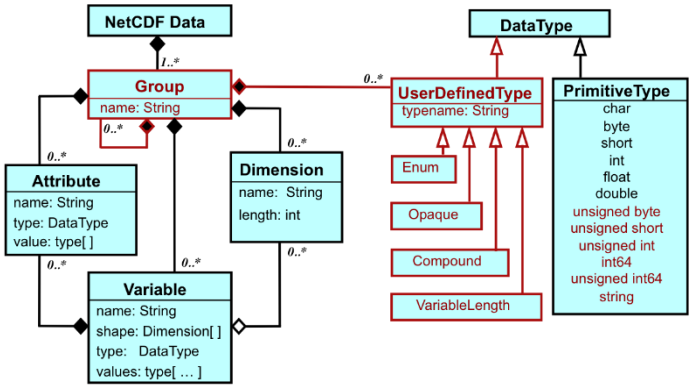
读取函数　① 从指定的文件中**读取多个变量的数据**　② 读取文件中**所有变量的数据**（返回一个字典）
③ 读取文件的**全局属性**　④ 读取**指定变量的局部属性**

5.2.2.1 读取多个变量的数据

```
def read_netcdf(file_path,variable_names):
    results=[]
    ds = nc.Dataset(file_path, 'r')
    for var_name in variable_names:
        if var_name in ds.variables:
            print(var_name)
            data = ds.variables[var_name][:]    #读取全部数据
            results.append(np.copy(data))       #注意：创建一个副本
        else:
            ds.close()
            raise ValueError(f"Variable '{var_name}' not found in the NetCDF file.")
    ds.close()
    return np.array(results)
```

5.2.2.2 读取所有变量的数据

```
def read_all_variables(file_path):
    data_dict={}
    ds = nc.Dataset(file_path, 'r')
    for var_name in ds.variables:
        print("Variable name: ", var_name)
        data = ds.variables[var_name][:]
        data_dict[var_name] = np.copy(data)
    ds.close()
    return data_dict
```



5.2.2.3 读取全局属性

```
def read_global_attributes(file_path):
    global_attributes = {}
    ds = nc.Dataset(file_path, 'r')
    for attr_name in ds.__dict__:
        print("Attribute name: ", attr_name)
        global_attributes[attr_name] = ds.getncattr(attr_name)
    ds.close()
    return global_attributes
```

5.2.2.4 读取指定变量的局部属性

```
def read_variable_attributes(file_path, variable_name):
    variable_attributes = {}
    ds = nc.Dataset(file_path, 'r')
    if variable_name in ds.variables:
        for attr_name in ds.variables[variable_name].ncattrs():
            print("Attribute name: ", attr_name)
            variable_attributes[attr_name] =
ds.variables[variable_name].getncattr(attr_name)
        ds.close()
        return variable_attributes
    else:
        ds.close()
        raise ValueError(f"Variable '{variable_name}' not found in the NetCDF file.")
```