ModelArts

MoXing 开发指南

文档版本 01

发布日期 2023-04-11

华为技术有限公司

商标声明

HUAWEI和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 MoXing Framework 简介

MoXing Framework模块为MoXing提供基础公共组件，例如访问华为云的OBS服务，和具体的AI引擎解耦，在ModelArts支持的所有AI引擎(TensorFlow、MXNet、PyTorch、MindSpore等)下均可以使用。目前，提供的MoXing Framework功能中主要包含操作OBS组件，即下文中描述的mox.file接口。

为什么要用 mox.file

OBS服务是一个基于对象的海量存储服务，无法通过像访问unix本地文件系统那样访问OBS上的文件，必须通过网络请求读写文件。为了在ModelArts服务中，更便捷的访问OBS目录，建议通过MoXing Framework（mox.file）对OBS目录进行操作。

通常Python打开一个本地文件，如下所示：

```
with open('/tmp/a.txt', 'r') as f:
  print(f.read())
```

OBS目录通常以“obs://”开头，比如“obs://bucket/XXX.txt”。用户无法直接使用open方法打开OBS文件，上面描述的打开本地文件的代码将会报错。

OBS提供了很多方式和工具给用户使用，如SDK、API、console、OBS Browser等，ModelArts mox.file提供了一套更为方便的访问OBS的API，允许用户通过一系列模仿操作本地文件系统的API来操作OBS文件。例如，可以使用以下代码来打开一个OBS上的文件。

```
import moxing as mox
with mox.file.File('obs://bucket_name/a.txt', 'r') as f:
  print(f.read())
```

例如，列举一个本地路径会使用如下Python代码。

```
import os
os.listdir('/tmp/my_dir/')
```

如果想要列举一个OBS路径，mox.file则需要如下代码：

```
import moxing as mox
mox.file.list_directory('obs://bucket_name/my_dir/')
```

2 快速入门

本文档介绍如何在ModelArts中调用MoXing Framework接口。

进入 ModelArts，创建 Notebook 实例

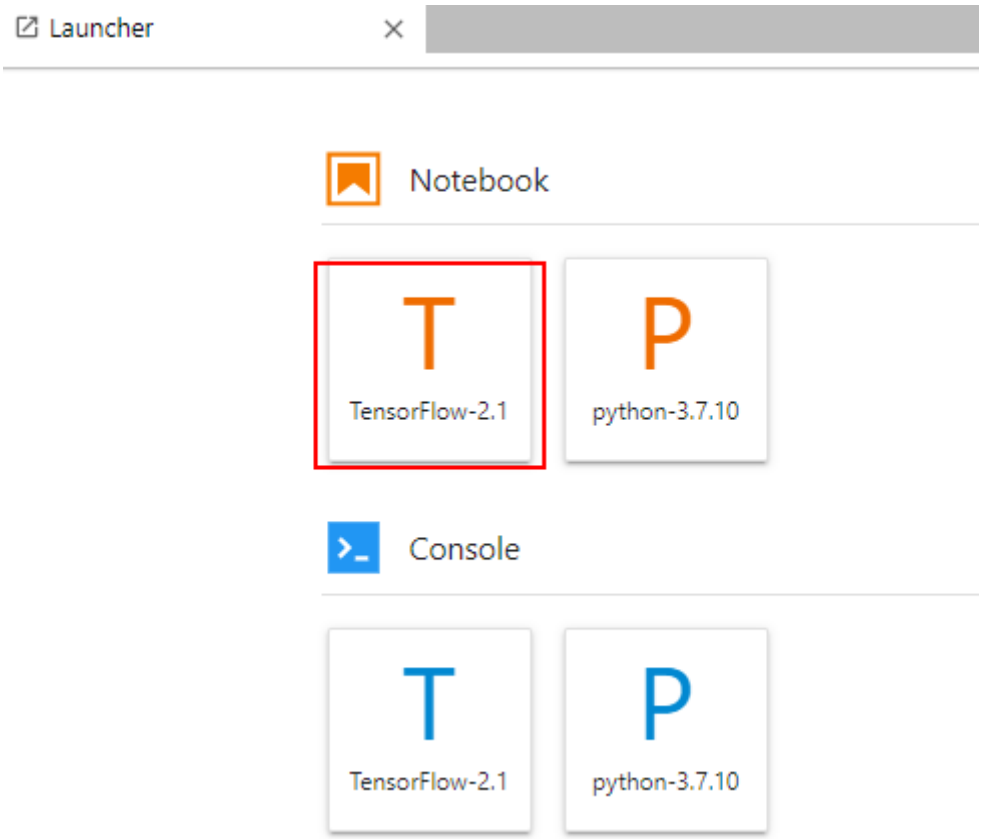
1. 登录ModelArts管理控制台，在左侧菜单栏中选择“开发环境>Notebook”，进入“Notebook”管理页面。

   图 2-1 进入 Notebook

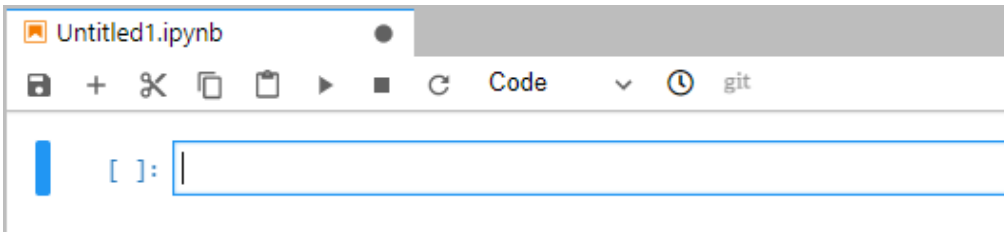
2. 单击“创建”进入“创建Notebook”页面，参考**创建Notebook实例**填写信息并完成Notebook实例创建。
3. 当Notebook实例创建完成后，且状态为“运行中”时，单击Notebook名称，或者单击“操作”列中的“打开”，进入“JupyterLab Notebook”开发页面。
4. 在JupyterLab的“Launcher”页签下，以TensorFlow为例，您可以单击TensorFlow，创建一个用于编码的文件。

图 2-2 选择不同的 AI 引擎



文件创建完成后，系统默认进入“JupyterLab”编码页面。

图 2-3 进入编码页面



调用 mox.file

输入如下代码，实现如下几个简单的功能。

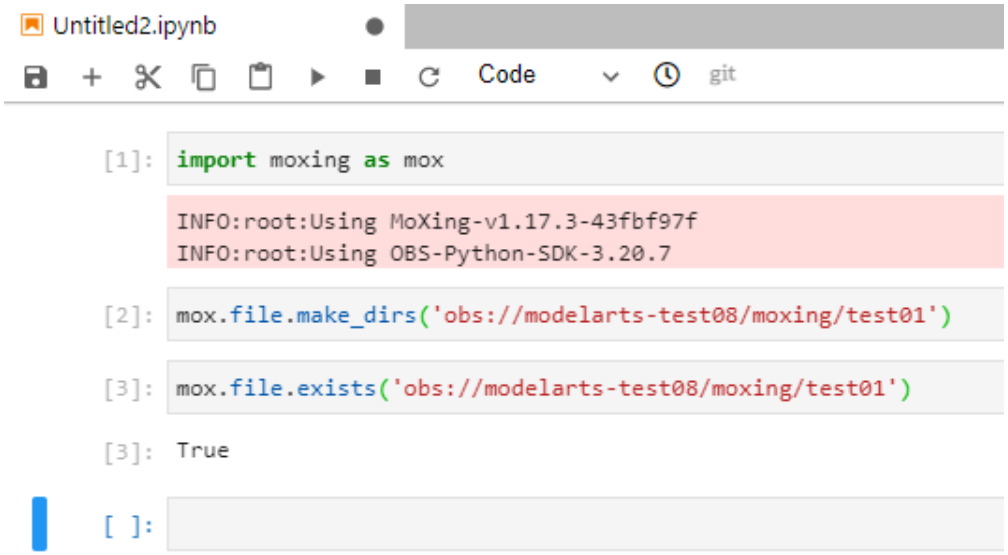
1. 引入MoXing Framework。
2. 在已有的“modelarts-test08/moxing”目录下，创建一个“test01”文件夹。
3. 调用代码检查“test01”文件夹是否存在，如果存在，表示上一个操作已成功。

```
import moxing as mox

mox.file.make_dirs('obs://modelarts-test08/moxing/test01')
mox.file.exists('obs://modelarts-test08/moxing/test01')
```

执行结果如图2-4所示。注意，每输入一行代码，单击下“Run”运行。您也可以进入OBS管理控制台，检查“modelarts-test08/moxing”目录，查看“test01”文件夹是否已创建成功。更多MoXing的常用操作请参见常用操作的样例代码。

图 2-4 运行示例



3 引入 MoXing Framework 模块

使用MoXing Framework前，您需要在代码的开头先引入MoXing Framework模块。

引入 MoXing Framework

执行如下代码，引入MoXing模块。

```
import moxing as mox
```

当引入MoXing+AI引擎相关的模块时，会涵盖所有Framework的功能，例如如下操作，这里的mox同时涵盖了所有moxing.tensorflow和moxing.framework下的所有API。

```
import moxing.tensorflow as mox
```

引入 MoXing Framework 的相关说明

在引入MoXing模块后，Python的标准logging模块会被设置为INFO级别，并打印打印版本号信息。可以通过以下API重新设置logging的等级。

```
import logging

from moxing.framework.util import runtime
runtime.reset_logger(level=logging.WARNING)
```

可以在引入moxing之前，配置环境变量MOX_SILENT_MODE=1，来防止MoXing打印版本号。使用如下Python代码来配置环境变量，需要在import moxing之前就将环境变量配置好。

```
import os
os.environ['MOX_SILENT_MODE'] = '1'
import moxing as mox
```

4 mox.file 与本地接口的对应关系和切换

API 对应关系

- Python：指本地使用Python对本地文件的操作接口。支持一键切换为对应的MoXing文件操作接口（mox.file）。
- mox.file：指MoXing框架中用于文件操作的接口，其与python接口一一对应关系。
- tf.gfile：指MoXing文件操作接口一一对应的TensorFlow相同功能的接口，在MoXing中，无法自动将文件操作接口自动切换为TensorFlow的接口，下表呈现内容仅表示功能类似，帮助您更快速地了解MoXing文件操作接口的功能。

表 4-1 API 对应关系

Python（本地文件操作接口）	mox.file（MoXing文件操作接口）	tf.gfile（TensorFlow文件操作接口）
glob.glob	mox.file.glob	tf.gfile.Glob
os.listdir	mox.file.list_directory(..., recursive=False)	tf.gfile.ListDirectory
os.makedirs	mox.file.make_dirs	tf.gfile.MakeDirs
os.mkdir	mox.file.mk_dir	tf.gfile.MkDir
os.path.exists	mox.file.exists	tf.gfile.Exists
os.path.getsize	mox.file.get_size	-
os.path.isdir	mox.file.is_directory	tf.gfile.IsDirectory
os.remove	mox.file.remove(..., recursive=False)	tf.gfile.Remove
os.rename	mox.file.rename	tf.gfile.Rename
os.scandir	mox.file.scan_dir	-
os.stat	mox.file.stat	tf.gfile.Stat
os.walk	mox.file.walk	tf.gfile.Walk

Python（本地文件操作接口）	mox.file（MoXing文件操作接口）	tf.gfile（TensorFlow文件操作接口）
open	mox.file.File	tf.gfile.FastGFile(tf.gfile.Gfile)
shutil.copyfile	mox.file.copy	tf.gfile.Copy
shutil.copytree	mox.file.copy_parallel	-
shutil.rmtree	mox.file.remove(..., recursive=True)	tf.gfile.DeleteRecursively

一键切换

您可以通过一行代码，将表4-1中OS的API映射到mox.file下。将以下代码写到启动脚本的最前面，在之后的Python运行中，当调用表格第一列的OS相关的API时，会自动映射到第二列mox.file的API。

```
import moxing as mox
mox.file.shift('os', 'mox')
```

在做完shift之后，可以直接通过os.listdir或者open方法操作OBS目录或文件，样例代码如下所示。

```
import os
import moxing as mox

mox.file.shift('os', 'mox')

print(os.listdir('obs://bucket_name'))
with open('obs://bucket_name/hello_world.txt') as f:
  print(f.read())
```

5 常用操作的样例代码

读写操作

- 读取一个OBS文件。

  例如读取“obs://bucket_name/obs_file.txt”文件内容，返回string（字符串类型）。

  ```
  import moxing as mox
  file_str = mox.file.read('obs://bucket_name/obs_file.txt')
  ```

  也可以使用打开文件对象并读取的方式来实现，两者是等价的。

  ```
  import moxing as mox
  with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
    file_str = f.read()
  ```

- 从文件中读取一行，返回string，以换行符结尾。同样可以打开OBS的文件对象。

  ```
  import moxing as mox
  with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
    file_line = f.readline()
  ```

- 从文件中读取所有行，返回一个list，每个元素是一行，以换行符结尾。

  ```
  import moxing as mox
  with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
    file_line_list = f.readlines()
  ```

- 以二进制模式读取一个OBS文件。

  例如读取“obs://bucket_name/obs_file.bin”文件内容，返回bytes（字节类型）。

  ```
  import moxing as mox
  file_bytes = mox.file.read('obs://bucket_name/obs_file.bin', binary=True)
  ```

  也可以使用打开文件对象并读取的方式来实现，两者是等价的。

  ```
  import moxing as mox
  with mox.file.File('obs://bucket_name/obs_file.bin', 'rb') as f:
    file_bytes = f.read()
  ```

  以二进制模式打开的文件也支持读取一行或者读取所有行，用法不变。

- 将字符串写入一个文件。

  例如将字符串“Hello World!”写入OBS文件“obs://bucket_name/obs_file.txt”中。

  ```
  import moxing as mox
  mox.file.write('obs://bucket_name/obs_file.txt', 'Hello World!')
  ```

  也可以使用打开文件对象并写入的方式来实现，两者是等价的。

  ```
  import moxing as mox
  with mox.file.File('obs://bucket_name/obs_file.txt', 'w') as f:
    f.write('Hello World!')
  ```

说明

用写入模式打开文件或者调用mox.file.write时，如果被写入文件不存在，则会创建，如果已经存在，则会覆盖。

- 追加一个OBS文件。

  例如将字符串“Hello World!”追加到“obs://bucket_name/obs_file.txt”文件中。

```
import moxing as mox
mox.file.append('obs://bucket_name/obs_file.txt', 'Hello World!')
```

  也可以使用打开文件对象并追加的方式来实现，两者是等价的。

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'a') as f:
  f.write('Hello World!')
```

  用追加模式打开文件或者调用**mox.file.append**时，如果被写入文件不存在，则会创建，如果已经存在，则直接追加。

  当被追加的源文件比较大时，例如“obs://bucket_name/obs_file.txt”文件大小超过5MB时，追加一个OBS文件的性能比较低。

说明

如果以写入模式(w)或追加模式(a)打开文件，当调用write方法时，待写入内容只是暂时的被存在的缓冲区，直到关闭文件对象(退出with语句时会自动关闭文件对象），或者主动调用文件对象的close()方法或flush()方法，文件内容才会被写入。

列举操作

- 列举一个OBS目录，只返回顶层结果（相对路径），不做递归列举。

  例如列举“obs://bucket_name/object_dir”，返回该目录下所有的文件和文件夹，不会递归查询。

  假设“obs://bucket_name/object_dir”中有如下结构

```
bucket_name
    |- object_dir
      |- dir0
        |- file00
      |- file1
```

  调用如下代码：

```
import moxing as mox
mox.file.list_directory('obs://bucket_name/object_dir')
```

  返回一个list：

```
['dir0', 'file1']
```

- 递归列举一个OBS目录，返回目录中所有的文件和文件夹（相对路径），并且会做递归查询。

  假设obs://bucket_name/object_dir中有如下结构。

```
bucket_name
    |- object_dir
      |- dir0
        |- file00
      |- file1
```

  调用如下代码：

```
import moxing as mox
mox.file.list_directory('obs://bucket_name/object_dir', recursive=True)
```

  返回一个list：

```
['dir0', 'dir0/file00', 'file1']
```

创建文件夹操作

创建一个OBS目录（即OBS文件夹）。支持递归创建，即如果“sub_dir_0”文件夹不存在，也会自动被创建，如果已经存在，则不起任何作用。

```
import moxing as mox
mox.file.make_dirs('obs://bucket_name/sub_dir_0/sub_dir_1')
```

查询操作

- 判断一个OBS文件是否存在，如果存在则返回**True**，如果不存在则返回**False**。
```
import moxing as mox
mox.file.exists('obs://bucket_name/sub_dir_0/file.txt')
```
- 判断一个OBS文件夹是否存在，如果存在则返回**True**，如果不存在则返回**False**。
```
import moxing as mox
mox.file.exists('obs://bucket_name/sub_dir_0/sub_dir_1')
```

说明

由于OBS允许同名的文件和文件夹（Unix操作系统不允许），如果存在同名的文件和文件夹，例如“obs://bucket_name/sub_dir_0/abc”，当调用mox.file.exists时，不论abc是文件还是文件夹，都会返回True。

- 判断一个OBS路径是否为文件夹，如果是则返回**True**，否则返回**False**。
```
import moxing as mox
mox.file.is_directory('obs://bucket_name/sub_dir_0/sub_dir_1')
```

说明

由于OBS允许同名的文件和文件夹（Unix操作系统不允许），如果存在同名的文件和文件夹，例如obs://bucket_name/sub_dir_0/abc，当调用mox.file.is_directory时，会返回True。

- 获取一个OBS文件的大小，单位为bytes。

  例如获取“obs://bucket_name/obs_file.txt”的文件大小。
```
import moxing as mox
mox.file.get_size('obs://bucket_name/obs_file.txt')
```
- 递归获取一个OBS文件夹下所有文件的大小，单位为bytes。

  例如获取“obs://bucket_name/object_dir”目录下所有文件大小的总和。
```
import moxing as mox
mox.file.get_size('obs://bucket_name/object_dir', recursive=True)
```
- 获取一个OBS文件或文件夹的stat信息，stat信息中包含如下信息。
  - length：文件大小。
  - mtime_nsec：创建时间戳。
  - is_directory：是否为目录。

  例如查询一个OBS文件“obs://bucket_name/obs_file.txt”，此文件地址也可以替换成一个文件夹地址。
```
import moxing as mox
stat = mox.file.stat('obs://bucket_name/obs_file.txt')
print(stat.length)
print(stat.mtime_nsec)
print(stat.is_directory)
```

删除操作

- 删除一个OBS文件。

  例如删除“obs://bucket_name/obs_file.txt”。
```
import moxing as mox
mox.file.remove('obs://bucket_name/obs_file.txt')
```

- 删除一个OBS目录，并且递归的删除这个目录下的所有内容。如果这个目录不存在，则会报错。

  例如删除“obs://bucket_name/sub_dir_0”下的所有内容。
  ```
  import moxing as mox
  mox.file.remove('obs://bucket_name/sub_dir_0', recursive=True)
  ```

移动和拷贝操作

- 移动一个OBS文件或文件夹。移动操作本身是用“拷贝+删除”来实现的。
  - 一个OBS文件移动到另一个OBS文件，例如将“obs://bucket_name/obs_file.txt”移动到“obs://bucket_name/obs_file_2.txt”。
    ```
    import moxing as mox
    mox.file.rename('obs://bucket_name/obs_file.txt', 'obs://bucket_name/obs_file_2.txt')
    ```

    说明

    移动和拷贝操作不可以跨桶，必须在同一个桶内操作。
  - 从OBS移动到本地，例如将“obs://bucket_name/obs_file.txt”移动到“/tmp/obs_file.txt”。
    ```
    import moxing as mox
    mox.file.rename('obs://bucket_name/obs_file.txt', '/tmp/obs_file.txt')
    ```
  - 从本地移动到OBS，例如将“/tmp/obs_file.txt”移动到“obs://bucket_name/obs_file.txt”。
    ```
    import moxing as mox
    mox.file.rename('/tmp/obs_file.txt', 'obs://bucket_name/obs_file.txt')
    ```
  - 从本地移动到本地，例如将“/tmp/obs_file.txt”移动到“/tmp/obs_file_2.txt”，该操作相当于**os.rename**。
    ```
    import moxing as mox
    mox.file.rename('/tmp/obs_file.txt', '/tmp/obs_file_2.txt')
    ```

  所有的移动操作均可以操作文件夹，如果操作的是文件夹，那么则会递归移动文件夹下所有的内容。
- 拷贝一个文件。**mox.file.copy**仅支持对文件操作，如果要对文件夹进行操作，请使用**mox.file.copy_parallel**。
  - 从OBS拷贝到OBS，例如将“obs://bucket_name/obs_file.txt”拷贝到“obs://bucket_name/obs_file_2.txt”。
    ```
    import moxing as mox
    mox.file.copy('obs://bucket_name/obs_file.txt', 'obs://bucket_name/obs_file_2.txt')
    ```
  - 将OBS文件拷贝到本地，也就是下载一个OBS文件。例如下载“obs://bucket_name/obs_file.txt”到本地“/tmp/obs_file.txt”。
    ```
    import moxing as mox
    mox.file.copy('obs://bucket_name/obs_file.txt', '/tmp/obs_file.txt')
    ```
  - 将本地文件拷贝到OBS，也就是上传一个OBS文件，例如上传“/tmp/obs_file.txt”到“obs://bucket_name/obs_file.txt”。
    ```
    import moxing as mox
    mox.file.copy('/tmp/obs_file.txt', 'obs://bucket_name/obs_file.txt')
    ```
  - 将本地文件拷贝到本地，操作等价于**shutil.copyfile**，例如将“/tmp/obs_file.txt”拷贝到“/tmp/obs_file_2.txt”。
    ```
    import moxing as mox
    mox.file.copy('/tmp/obs_file.txt', '/tmp/obs_file_2.txt')
    ```
- 拷贝一个文件夹。**mox.file.copy_parallel**仅支持对文件夹操作，如果要对文件进行操作，请使用**mox.file.copy**。
  - 从OBS拷贝到OBS，例如将obs://bucket_name/sub_dir_0拷贝到obs://bucket_name/sub_dir_1
    ```
    import moxing as mox
    mox.file.copy_parallel('obs://bucket_name/sub_dir_0', 'obs://bucket_name/sub_dir_1')
    ```

- 将OBS文件夹拷贝到本地，也就是下载一个OBS文件夹。例如下载“obs://bucket_name/sub_dir_0”到本地“/tmp/sub_dir_0”。
```
import moxing as mox
mox.file.copy_parallel('obs://bucket_name/sub_dir_0', '/tmp/sub_dir_0')
```
- 将本地文件夹拷贝到OBS，也就是上传一个OBS文件夹，例如上传“/tmp/sub_dir_0”到“obs://bucket_name/sub_dir_0”。
```
import moxing as mox
mox.file.copy_parallel('/tmp/sub_dir_0', 'obs://bucket_name/sub_dir_0')
```
- 将本地文件夹拷贝到本地，操作等价于**shutil.copytree**，例如将“/tmp/sub_dir_0”拷贝到“/tmp/sub_dir_1”。
```
import moxing as mox
mox.file.copy_parallel('/tmp/sub_dir_0', '/tmp/sub_dir_1')
```

6 进阶用法的样例代码

如果您已经熟悉了常用操作，同时熟悉MoXing Framework API文档以及常用的Python编码，您可以参考本章节使用MoXing Framework的一些进阶用法。

读取完毕后将文件关闭

当读取OBS文件时，实际调用的是HTTP连接读去网络流，注意要记得在读取完毕后将文件关闭。为了防止忘记文件关闭操作，推荐使用with语句，在with语句退出时会自动调用**mox.file.File**对象的**close()**方法：

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
  data = f.readlines()
```

利用 pandas 读或写一个 OBS 文件

- 利用pandas读一个OBS文件。
  ```
  import pandas as pd
  import moxing as mox
  with mox.file.File("obs://bucket_name/b.txt", "r") as f:
    csv = pd.read_csv(f)
  ```
- 利用pandas写一个OBS文件。
  ```
  import pandas as pd
  import moxing as mox
  df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
  with mox.file.File("obs://bucket_name/b.txt", "w") as f:
    df.to_csv(f)
  ```

利用文件对象读取图片

使用opencv打开一张图片时，无法传入一个obs路径，需要利用文件对象读取，考虑以下代码是无法读取到该图片的。

```
import cv2
cv2.imread('obs://bucket_name/xxx.jpg', cv2.IMREAD_COLOR)
```

修改为如下代码：

```
import cv2
import numpy as np
import moxing as mox
img = cv2.imdecode(np.fromstring(mox.file.read('obs://bucket_name/xxx.jpg', binary=True), np.uint8),
cv2.IMREAD_COLOR)
```

利用已有的接口先实现一个不被 mox.file 支持的接口

如果用户API中调用到了不被mox.file支持的接口，那么可以利用已有的接口先实现这个新接口，然后覆盖原API。如**os.path.isfile**不在支持的接口范围内，当用户调用**mox.file.shift('os', 'mox')**后，**os.path.isfile**调用的依然是Python的原生builtin方法，按如下代码将该方法覆盖：

```
import os
import moxing as mox

_origin_isfile = os.path.isfile

def _patch_isfile(path):
  return not mox.file.isdir(path)

setattr(os.path, 'isfile', _patch_isfile)
```

将一个不支持 obs 路径的 API 改造成支持 OBS 路径的 API

pandas中对h5的文件读写**to_hdf**和**read_hdf**既不支持OBS路径，也不支持输入一个文件对象，考虑以下代码会出现错误。

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]}, index=['a', 'b', 'c'])
df.to_hdf('obs://wolfros-net/hdftest.h5', key='df', mode='w')
pd.read_hdf('obs://wolfros-net/hdftest.h5')
```

通过重写pandas源码API的方式，将该API改造成支持OBS路径的形式。

- 写h5到OBS = 写h5到本地缓存 + 上传本地缓存到OBS + 删除本地缓存
- 从OBS读h5 = 下载h5到本地缓存 + 读取本地缓存 + 删除本地缓存

即将以下代码写在运行脚本的最前面，就能使运行过程中的**to_hdf**和**read_hdf**支持OBS路径。

```
import os
import moxing as mox
import pandas as pd
from pandas.io import pytables
from pandas.core.generic import NDFrame

to_hdf_origin = getattr(NDFrame, 'to_hdf')
read_hdf_origin = getattr(pytables, 'read_hdf')


def to_hdf_override(self, path_or_buf, key, **kwargs):
  tmp_dir = '/cache/hdf_tmp'
  file_name = os.path.basename(path_or_buf)
  mox.file.make_dirs(tmp_dir)
  local_file = os.path.join(tmp_dir, file_name)
  to_hdf_origin(self, local_file, key, **kwargs)
  mox.file.copy(local_file, path_or_buf)
  mox.file.remove(local_file)


def read_hdf_override(path_or_buf, key=None, mode='r', **kwargs):
  tmp_dir = '/cache/hdf_tmp'
  file_name = os.path.basename(path_or_buf)
  mox.file.make_dirs(tmp_dir)
  local_file = os.path.join(tmp_dir, file_name)
  mox.file.copy(path_or_buf, local_file)
  result = read_hdf_origin(local_file, key, mode, **kwargs)
  mox.file.remove(local_file)
  return result

setattr(NDFrame, 'to_hdf', to_hdf_override)
```

```
setattr(pytables, 'read_hdf', read_hdf_override)
setattr(pd, 'read_hdf', read_hdf_override)
```

利用 moxing，使 h5py.File 支持 OBS

```
import os
import h5py
import numpy as np
import moxing as mox

h5py_File_class = h5py.File

class OBSFile(h5py_File_class):
  def __init__(self, name, *args, **kwargs):
    self._tmp_name = None
    self._target_name = name
    if name.startswith('obs://'):
      self._tmp_name = name.replace('/', '_')
      if mox.file.exists(name):
        mox.file.copy(name, os.path.join('cache', 'h5py_tmp', self._tmp_name))
      name = self._tmp_name

    super(OBSFile, self).__init__(name, *args, **kwargs)

  def close(self):
    if self._tmp_name:
      mox.file.copy(self._tmp_name, self._target_name)

    super(OBSFile, self).close()


setattr(h5py, 'File', OBSFile)

arr = np.random.randn(1000)
with h5py.File('obs://bucket/random.hdf5', 'r') as f:
  f.create_dataset("default", data=arr)

with h5py.File('obs://bucket/random.hdf5', 'r') as f:
  print(f.require_dataset("default", dtype=np.float32, shape=(1000,)))
```