ModelArts

MoXing Developer Guide

Issue	01
Date	2023-04-11

HUAWEI

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: https://www.huawei.com

Email: support@huawei.com

Contents

1 Introduction to MoXing Framework

MoXing Framework provides basic common components for MoXing. For example, it can be used to access Object Storage Service (OBS) on HUAWEI CLOUD. It is decoupled from specific AI engines and can be used in all AI engines (TensorFlow, MXNet, PyTorch, and MindSpore) supported by ModelArts. Currently, MoXing Framework mainly involves the OBS operation component, that is, the mox.file APIs described in the following.

Why mox.file

OBS is an object-based massive storage service. It cannot access files on OBS in the same way as accessing a local UNIX file system. You must read and write files on OBS through network requests. You are advised to use MoXing Framework (mox.file) to perform operations on the OBS directory in ModelArts.

Generally, Python provides the following function to open a local file:

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

An OBS directory usually starts with **obs://**, for example, **obs://bucket/XXX.txt**. You cannot directly use the **open** function to open an OBS file. The preceding code for opening a local file will report an error.

OBS provides many functions and tools, such as SDK, API, OBS Console, and OBS Browser. ModelArts mox.file provides a set of APIs for accessing OBS. The APIs simulate the operations of a local file system, allowing users to operate OBS files conveniently. For example, you can use the following code to open a file on OBS:

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

The following Python code lists a local path:

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

To list an OBS path, you can use the following code of mox.file:

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

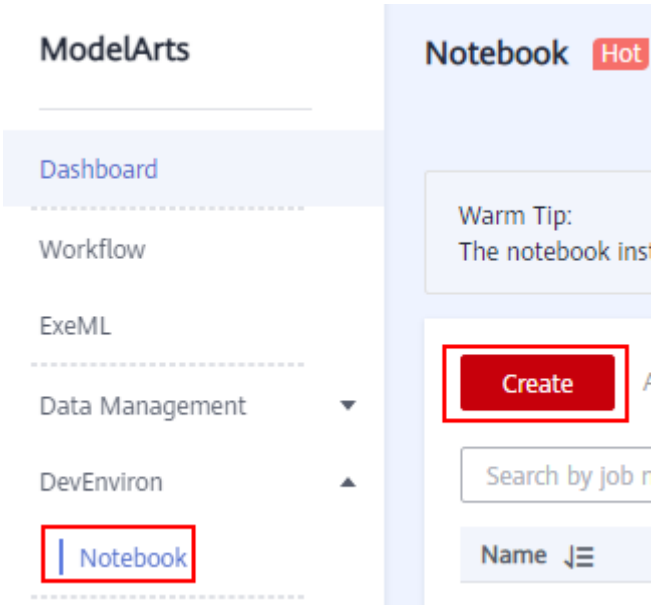

2 Getting Started

This document describes how to call MoXing Framework APIs in ModelArts.

Logging In to ModelArts and Creating a Notebook Instance

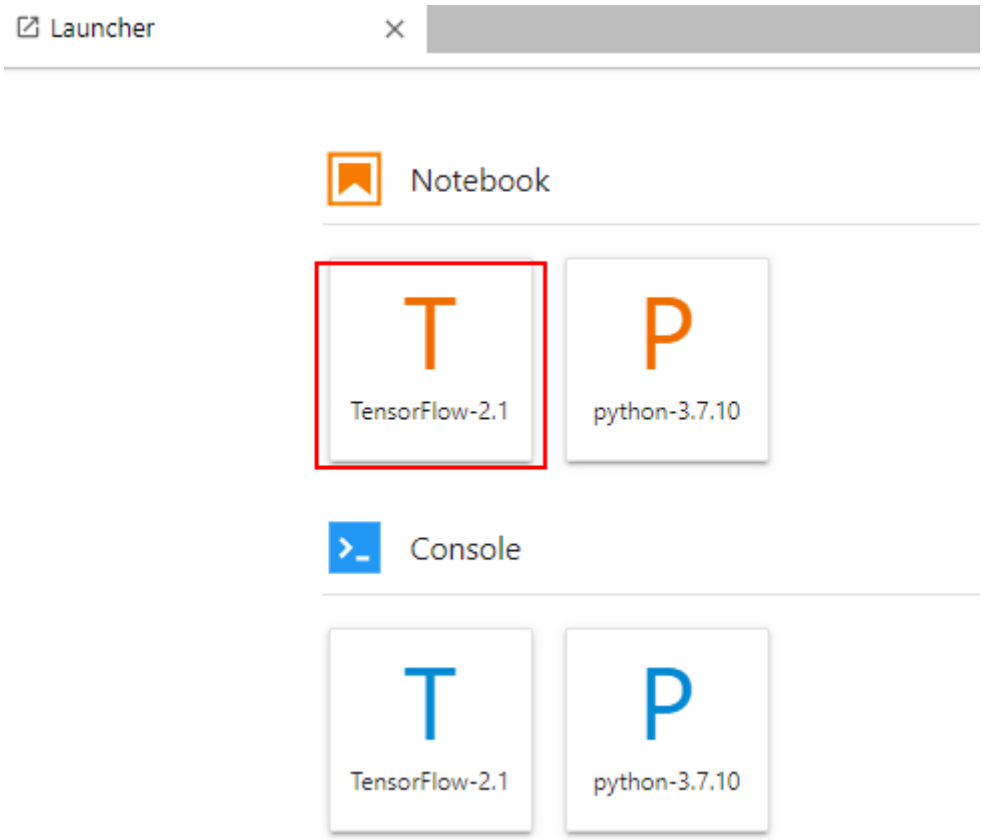
1. Log in to the ModelArts management console. In the left navigation pane, choose **DevEnviron** > **Notebook** to access the **Notebook** page.

   Figure 2-1 Accessing the Notebook page

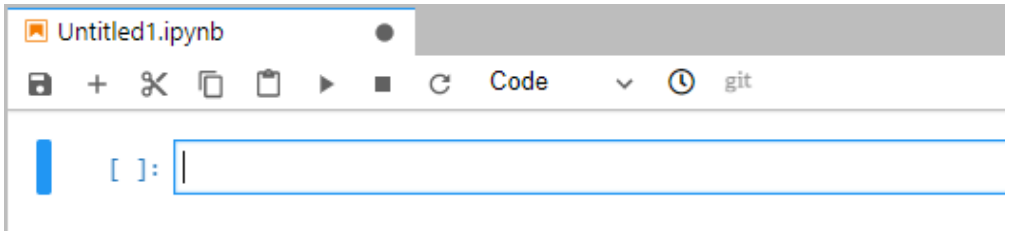
2. Click **Create**. On the **Create Notebook** page that is displayed, create a notebook instance by referring to **Creating a Notebook Instance**.
3. After a notebook instance is created and is in the **Running** status, click the notebook instance name or click **Open** in the **Operation** column to go to the **JupyterLab** development page.
4. On the **Launcher** tab page of JupyterLab, for example, click **TensorFlow** to create a file for encoding.

Figure 2-2 Selecting an AI engine



After the file is created, the **JupyterLab** page is displayed by default.

Figure 2-3 Encoding page



Calling mox.file.

Enter the following code to implement the following simple functions:

1. Introduce MoXing Framework.
2. Create the **test01** folder in the existing **modelarts-test08/moxing** directory.
3. Check whether the **test01** folder exists. If the folder exists, the preceding operation is successful.

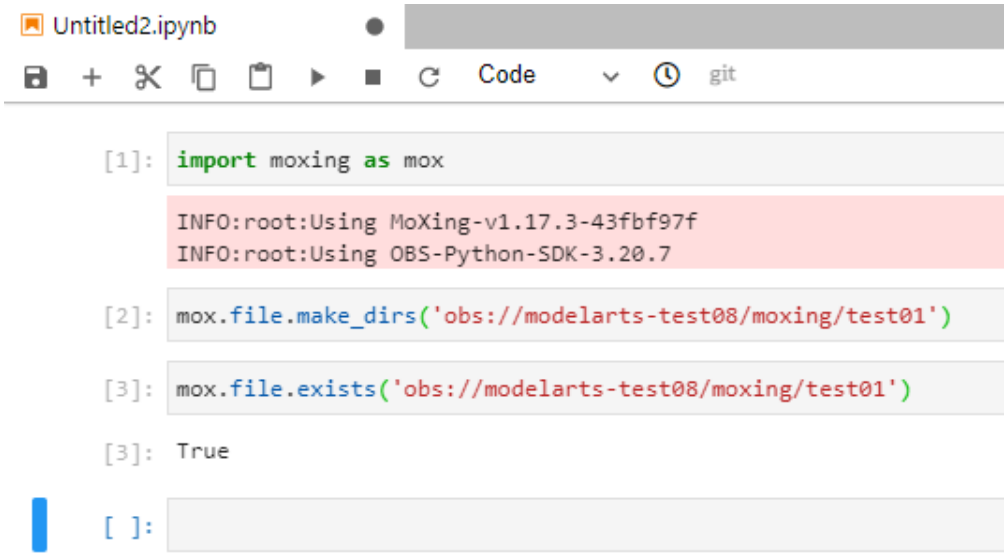
```
import moxing as mox

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

Figure 2-4 shows the result. Note that each time you enter a line of code, click **Run**. You can also go to OBS Console and check whether the **test01** folder has

been created in the **modelarts-test08/moxing** directory. For more common MoXing operations, see **Sample Code for Common Operations**.

Figure 2-4 Example



3 Introducing MoXing Framework

Before using MoXing Framework, you need to introduce the MoXing Framework module at the beginning of the code.

Introducing MoXing Framework

Run the following code to import the MoXing module:

```
import moxing as mox
```

After the MoXing module is introduced, all functions of MoXing Framework are covered. For example, in the following code, **mox** covers all APIs under moxing.tensorflow and moxing.framework:

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

Related Notes

After the MoXing module is introduced, the standard **logging** module of Python is set to the INFO level, and the version number is printed. You can use the following API to reset the **logging** level:

```
import logging

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

Before introducing MoXing, you can set the **MOX_SILENT_MODE** environment variable to **1** to prevent MoXing from printing the version number. Use the following Python code to configure the environment variable. You need to configure the environment variables before importing MoXing.

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


4 Mapping Between mox.file and Local APIs and Switchover

API Mapping

- Python: local file operation APIs of Python. The APIs can be shifted to the corresponding MoXing file operation APIs (mox.file) by one click.
- mox.file: file operation APIs of MoXing Framework. The APIs correspond to the Python APIs.
- tf.gfile: TensorFlow APIs with the same functions as MoXing file operation APIs. In MoXing, file operation APIs cannot be automatically switched to TensorFlow APIs. The following table lists only the APIs with similar functions.

Table 4-1 API mapping

Python (Local File Operation API)	mox.file (MoXing File Operation API)	tf.gfile (TensorFlow File Operation API)
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

Python (Local File Operation API)	mox.file (MoXing File Operation API)	tf.gfile (TensorFlow File Operation API)
os.walk	mox.file.walk	tf.gfile.Walk
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

One-Click Shift

You can use the following code to map the OS APIs in **Table 4-1** to the mox.file APIs. Write the following code at the beginning of the boot script. When the OS APIs in the first column of the table are called during Python running, the APIs are automatically mapped to the APIs in the second column.

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

After the **shift** operation is completed, you can use the **os.listdir** or **open** function to operate OBS directories or files. The sample code is as follows:

```
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 Sample Code for Common Operations

Data Reads and Writes

- Read an OBS file.

  For example, if you read the **obs://bucket_name/obs_file.txt** file, the content is returned as strings.

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

  You can also open the file object and read data from it. Both methods are the same.

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

- Read a line from a file. A string that ends with a newline character is returned. You can also open the file object in OBS.

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

- Read all lines from a file. A list is returned, in which each element is a line and ends with a newline character.

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

- Read an OBS file in binary mode.

  For example, if you read the **obs://bucket_name/obs_file.bin** file, the content is returned as bytes.

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

  You can also open the file object and read data from it. Both methods are the same.

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

  One or all lines in a file opened in binary mode can be read with the same method.

- Write a string to a file.

  For example, write **Hello World!** into the **obs://bucket_name/obs_file.txt** file.

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


You can also open the file object and write data into it. Both methods are the same.

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

NOTE

When you open a file in write mode or call **mox.file.write**, if the file to be written does not exist, the file will be created. If the file to be written already exists, the file is overwritten.

- Append content to an OBS file.

  For example, append **Hello World!** to the **obs://bucket_name/obs_file.txt** file.

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

  You can also open the file object and append content to it. Both methods are the same.

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

  When you open a file in append mode or call **mox.file.append**, if the file to be appended does not exist, the file will be created. If the file to be appended already exists, the content is directly appended.

  If the size of the source file to be appended is large, for example, the **obs://bucket_name/obs_file.txt** file exceeds 5 MB, the append performance is low.

  NOTE

  If the file object is opened in write (w) or append (a) mode, when the **write** function is called, the content to be written is temporarily stored in the cache until the file object is closed (the file object is automatically closed when the **with** statement exits). Alternatively, you can call the **close()** or **flush()** function of the file object to write the file content.

List

- List an OBS directory. Only the top-level result (relative path) is returned. Recursive listing is not performed.

  For example, if you list **obs://bucket_name/object_dir**, all files and folders in the directory are returned, but recursive queries are not performed.

  Assume that **obs://bucket_name/object_dir** is in the following structure:

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

  Call the following code:

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

  The following list is returned:

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

- Recursively list an OBS directory. All files and folders (relative paths) in the directory are returned, and recursive queries are performed.

  Assume that **obs://bucket_name/object_dir** is in the following structure:


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

Call the following code:

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

The following list is returned:

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

Create a Folder

Create an OBS directory, that is, an OBS folder. Recursive creation is supported. That is, if the **sub_dir_0** folder does not exist, it is automatically created. If the **sub_dir_0** folder exists, no folder will be created.

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

Query

- Check whether an OBS file exists. If the file exists, **True** is returned. If the file does not exist, **False** is returned.
  ```
  import moxing as mox
  mox.file.exists('obs://bucket_name/sub_dir_0/file.txt')
  ```
- Check whether an OBS folder exists. If the folder exists, **True** is returned. If the folder does not exist, **False** is returned.
  ```
  import moxing as mox
  mox.file.exists('obs://bucket_name/sub_dir_0/sub_dir_1')
  ```

  NOTE

  OBS allows files and folders with the same name exist (not allowed in UNIX). If a file or folder with the same name exists, for example, **obs://bucket_name/sub_dir_0/abc**, when **mox.file.exists** is called, **True** is returned regardless of whether **abc** is a file or folder.
- Check whether an OBS path is a folder. If it is a folder, **True** is returned. If it is not a folder, **False** is returned.
  ```
  import moxing as mox
  mox.file.is_directory('obs://bucket_name/sub_dir_0/sub_dir_1')
  ```

  NOTE

  OBS allows files and folders with the same name exist (not allowed in UNIX). If a file or folder with the same name exists, for example, **obs://bucket_name/sub_dir_0/abc**, when **mox.file.is_directory** is called, **True** is returned.
- Obtain the size of an OBS file, in bytes.

  For example, obtain the size of **obs://bucket_name/obs_file.txt**.
  ```
  import moxing as mox
  mox.file.get_size('obs://bucket_name/obs_file.txt')
  ```
- Recursively obtain the size of all files in an OBS folder, in bytes.

  For example, obtain the total size of all files in the **obs://bucket_name/object_dir** directory.
  ```
  import moxing as mox
  mox.file.get_size('obs://bucket_name/object_dir', recursive=True)
  ```
- Obtain the **stat** information about an OBS file or folder. The **stat** information contains the following:

  - **length**: file size
  - **mtime_nsec**: creation timestamp
  - **is_directory**: whether the path is a folder

  For example, if you want to query the OBS file **obs://bucket_name/obs_file.txt**, you can replace the file path with a folder path.

```
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)
```

Delete

- Delete an OBS file.

  For example, delete **obs://bucket_name/obs_file.txt**.

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

- Delete an OBS folder and recursively delete all content in the folder. If the folder does not exist, an error is reported.

  For example, delete all content in **obs://bucket_name/sub_dir_0**.

```
import moxing as mox
mox.file.remove('obs://bucket_name/sub_dir_0', recursive=True)
```

Move and Copy

- Move an OBS file or folder. The move operation is implemented by copying and deleting data.
  - Move an OBS file to another OBS file. For example, move **obs://bucket_name/obs_file.txt** to **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')
```

    NOTE

    The move and copy operation must be performed in the same bucket.

  - Move an OBS file to a local file. For example, move **obs://bucket_name/obs_file.txt** to **/tmp/obs_file.txt**.

```
import moxing as mox
mox.file.rename('obs://bucket_name/obs_file.txt', '/tmp/obs_file.txt')
```

  - Move a local file to an OBS file. For example, move **/tmp/obs_file.txt** to **obs://bucket_name/obs_file.txt**.

```
import moxing as mox
mox.file.rename('/tmp/obs_file.txt', 'obs://bucket_name/obs_file.txt')
```

  - Move a local file to another local file. For example, move **/tmp/obs_file.txt** to **/tmp/obs_file_2.txt**. This operation is equivalent to **os.rename**.

```
import moxing as mox
mox.file.rename('/tmp/obs_file.txt', '/tmp/obs_file_2.txt')
```

  You can move folders in the same way. If you move a folder, all content in the folder is moved recursively.

- Copy a file. **mox.file.copy** can be used to perform operations only on files. To perform operations on folders, use **mox.file.copy_parallel**.
  - Copy an OBS file to another OBS file. For example, copy **obs://bucket_name/obs_file.txt** to **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')
```

  - Copy an OBS file to a local file, that is, download an OBS file. For example, download **obs://bucket_name/obs_file.txt** to **/tmp/obs_file.txt**.

    ```
    import moxing as mox
    mox.file.copy('obs://bucket_name/obs_file.txt', '/tmp/obs_file.txt')
    ```

  - Copy a local file to an OBS file, that is, upload an OBS file. For example, upload **/tmp/obs_file.txt** to **obs://bucket_name/obs_file.txt**.

    ```
    import moxing as mox
    mox.file.copy('/tmp/obs_file.txt', 'obs://bucket_name/obs_file.txt')
    ```

  - Copy a local file to another local file. This operation is equivalent to **shutil.copyfile**. For example, copy **/tmp/obs_file.txt** to **/tmp/obs_file_2.txt**.

    ```
    import moxing as mox
    mox.file.copy('/tmp/obs_file.txt', '/tmp/obs_file_2.txt')
    ```

- Copy a folder. **mox.file.copy_parallel** can be used to perform operations only on folders. To perform operations on files, use **mox.file.copy**.
  - Copy an OBS folder to another OBS folder. For example, copy **obs://bucket_name/sub_dir_0** to **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')
    ```

  - Copy an OBS folder to a local folder, that is, download an OBS folder. For example, download **obs://bucket_name/sub_dir_0** to **/tmp/sub_dir_0**.

    ```
    import moxing as mox
    mox.file.copy_parallel('obs://bucket_name/sub_dir_0', '/tmp/sub_dir_0')
    ```

  - Copy a local folder to an OBS folder, that is, upload an OBS folder. For example, upload **/tmp/sub_dir_0** to **obs://bucket_name/sub_dir_0**.

    ```
    import moxing as mox
    mox.file.copy_parallel('/tmp/sub_dir_0', 'obs://bucket_name/sub_dir_0')
    ```

  - Copy a local folder to another local folder. This operation is equivalent to **shutil.copytree**. For example, copy **/tmp/sub_dir_0** to **/tmp/sub_dir_1**.

    ```
    import moxing as mox
    mox.file.copy_parallel('/tmp/sub_dir_0', '/tmp/sub_dir_1')
    ```


6 Sample Code for Advanced Applications

If you are familiar with common operations, the MoXing Framework API document, and common Python code, you can refer to this section to use advanced MoXing Framework functions.

Closing a File After File Reading Is Completed

When an OBS file is read, an HTTP connection is called to read the network stream. You need to close the file after the file is read. To prevent you from forgetting to close a file, you are advised to use the **with** statement. When the **with** statement exits, the **close()** function of the **mox.file.File** object is automatically called.

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

Reading or Writing an OBS File Using pandas

- Use **pandas** to read an OBS file.

  ```
  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)
  ```

- Use **pandas** to write an OBS file.

  ```
  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)
  ```

Reading an Image Using a File Object

When OpenCV is used to open an image, the OBS path cannot be passed and the image must be read using a file object. The following code cannot read the image:

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

Modify the code as follows:

```
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)
```

Using an Existing API to Implement an API That Is Not Supported by mox.file

To call an API that is not supported by mox.file, you can use an existing API to implement the API and overwrite the original API. For example, **os.path.isfile** is not supported by mox.file. After **mox.file.shift('os', 'mox')** is called, **os.path.isfile** still calls the native **builtin** function of Python. You can overwrite the function with the following code:

```
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)
```

Reconstructing an API That Does Not Support OBS Paths to an API That Supports OBS Paths

In **pandas**, **to_hdf** and **read_hdf** used to read and write H5 files do not support OBS paths, nor do they support file objects to be entered. The following code may cause errors:

```
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')
```

The API compiled using the pandas source code is rewritten to support OBS paths.

- Write H5 to OBS = Write H5 to the local cache + Upload the local cache to OBS + Delete the local cache
- Read H5 from OBS = Download H5 to the local cache + Read the local cache + Delete the local cache

That is, write the following code at the beginning of the script to enable **to_hdf** and **read_hdf** to support OBS paths:

```
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)
```

Use MoXing to Enable h5py.File to Support 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,)))
```

