pyarrow Documentation

Release

Apache Arrow Team

May 07, 2017

Arrow is a columnar in-memory analytics layer designed to accelerate big data. It houses a set of canonical in-memory representations of flat and hierarchical data along with multiple language-bindings for structure manipulation. It also provides IPC and common algorithm implementations.

This is the documentation of the Python API of Apache Arrow. For more details on the format and other language bindings see the main page for Arrow. Here will we only detail the usage of the Python API for Arrow and the leaf libraries that add additional functionality such as reading Apache Parquet files into Arrow structures.

CHAPTER 1

Install PyArrow

Conda

To install the latest version of PyArrow from conda-forge using conda:

```
conda install -c conda-forge pyarrow
```

Pip

Install the latest version from PyPI:

```
pip install pyarrow
```

Note: Currently there are only binary artifacts available for Linux and MacOS. Otherwise this will only pull the python sources and assumes an existing installation of the C++ part of Arrow. To retrieve the binary artifacts, you'll need a recent `pip` version that supports features like the `manylinux1` tag.

Installing from source

See *Development*.

CHAPTER 2

Development

Developing with conda

Linux and macOS

System Requirements

On macOS, any modern XCode (6.4 or higher; the current version is 8.3.1) is sufficient.

On Linux, for this guide, we recommend using gcc 4.8 or 4.9, or clang 3.7 or higher. You can check your version by running

```
$ gcc --version
```

On Ubuntu 16.04 and higher, you can obtain gcc 4.9 with:

```
$ sudo apt-get install g++-4.9
```

Finally, set gcc 4.9 as the active compiler using:

```
export CC=gcc-4.9
export CXX=g++-4.9
```

Environment Setup and Build

First, let's create a conda environment with all the C++ build and Python dependencies from conda-forge:

```
conda create -y -q -n pyarrow-dev \
      python=3.6 numpy six setuptools cython pandas pytest \
      cmake flatbuffers rapidjson boost-cpp thrift-cpp snappy zlib \
      brotli jemalloc -c conda-forge
source activate pyarrow-dev
```

Now, let's clone the Arrow and Parquet git repositories:

```
mkdir repos
cd repos
git clone https://github.com/apache/arrow.git
git clone https://github.com/apache/parquet-cpp.git
```

You should now see

```
$ ls -l
total 8
drwxrwxr-x 12 wesm wesm 4096 Apr 15 19:19 arrow/
drwxrwxr-x 12 wesm wesm 4096 Apr 15 19:19 parquet-cpp/
```

We need to set some environment variables to let Arrow's build system know about our build toolchain:

```
export ARROW_BUILD_TYPE=release

export ARROW_BUILD_TOOLCHAIN=$CONDA_PREFIX
export PARQUET_BUILD_TOOLCHAIN=$CONDA_PREFIX
export ARROW_HOME=$CONDA_PREFIX
export PARQUET_HOME=$CONDA_PREFIX
```

Now build and install the Arrow C++ libraries:

```
mkdir arrow/cpp/build
pushd arrow/cpp/build

cmake -DCMAKE_BUILD_TYPE=$ARROW_BUILD_TYPE \
      -DCMAKE_INSTALL_PREFIX=$ARROW_HOME \
      -DARROW_PYTHON=on \
      -DARROW_BUILD_TESTS=OFF \
      ..
make -j4
make install
popd
```

Now, optionally build and install the Apache Parquet libraries in your toolchain:

```
mkdir parquet-cpp/build
pushd parquet-cpp/build

cmake -DCMAKE_BUILD_TYPE=$ARROW_BUILD_TYPE \
      -DCMAKE_INSTALL_PREFIX=$PARQUET_HOME \
      -DPARQUET_BUILD_BENCHMARKS=off \
      -DPARQUET_BUILD_EXECUTABLES=off \
      -DPARQUET_ZLIB_VENDORED=off \
      -DPARQUET_BUILD_TESTS=off \
      ..

make -j4
make install
popd
```

Now, build pyarrow:

```
cd arrow/python
python setup.py build_ext --build-type=$ARROW_BUILD_TYPE \
       --with-parquet --with-jemalloc --inplace
```

If you did not build parquet-cpp, you can omit `--with-parquet`.

You should be able to run the unit tests with:

```
$ py.test pyarrow
============================= test session starts ====================
platform linux -- Python 3.6.1, pytest-3.0.7, py-1.4.33, pluggy-0.4.0
rootdir: /home/wesm/arrow-clone/python, inifile:
collected 198 items
```

```
pyarrow/tests/test_array.py ..........
pyarrow/tests/test_convert_builtin.py ..................
pyarrow/tests/test_convert_pandas.py ..........................
pyarrow/tests/test_feather.py ........................
pyarrow/tests/test_hdfs.py sssssssssssss
pyarrow/tests/test_io.py ................
pyarrow/tests/test_ipc.py ........
pyarrow/tests/test_jemalloc.py ss
pyarrow/tests/test_parquet.py ................
pyarrow/tests/test_scalars.py ..........
pyarrow/tests/test_schema.py .........
pyarrow/tests/test_table.py ............
pyarrow/tests/test_tensor.py ..............

====================== 181 passed, 17 skipped in 0.98 seconds ===========
```

Windows

First, make sure you can build the C++ library.

Now, we need to build and install the C++ libraries someplace.

```
mkdir cpp\build
cd cpp\build
set ARROW_HOME=C:\thirdparty
cmake -G "Visual Studio 14 2015 Win64" ^
      -DCMAKE_INSTALL_PREFIX=%ARROW_HOME% ^
      -DCMAKE_BUILD_TYPE=Release ^
      -DARROW_BUILD_TESTS=off ^
      -DARROW_PYTHON=on ..
cmake --build . --target INSTALL --config Release
cd ..\..
```

After that, we must put the install directory's bin path in our `%PATH%`:

```
set PATH=%ARROW_HOME%\bin;%PATH%
```

Now, we can build pyarrow:

```
cd python
python setup.py build_ext --inplace
```

Running C++ unit tests with Python

Getting `python-test.exe` to run is a bit tricky because your `%PYTHONPATH%` must be configured given the active conda environment:

```
set CONDA_ENV=C:\Users\wesm\Miniconda\envs\arrow-test
set PYTHONPATH=%CONDA_ENV%\Lib;%CONDA_ENV%\Lib\site-packages;%CONDA_ENV%\python35.zip;%CONDA_ENV%\DL
```

Now `python-test.exe` or simply `ctest` (to run all tests) should work.

CHAPTER 3

Pandas Interface

To interface with Pandas, PyArrow provides various conversion routines to consume Pandas structures and convert back to them.

DataFrames

The equivalent to a Pandas DataFrame in Arrow is a `pyarrow.table.Table`. Both consist of a set of named columns of equal length. While Pandas only supports flat columns, the Table also provides nested columns, thus it can represent more data than a DataFrame, so a full conversion is not always possible.

Conversion from a Table to a DataFrame is done by calling `pyarrow.table.Table.to_pandas()`. The inverse is then achieved by using `pyarrow.Table.from_pandas()`. This conversion routine provides the convience parameter `timestamps_to_ms`. Although Arrow supports timestamps of different resolutions, Pandas only supports nanosecond timestamps and most other systems (e.g. Parquet) only work on millisecond timestamps. This parameter can be used to already do the time conversion during the Pandas to Arrow conversion.

```
import pyarrow as pa
import pandas as pd

df = pd.DataFrame({"a": [1, 2, 3]})
# Convert from Pandas to Arrow
table = pa.Table.from_pandas(df)
# Convert back to Pandas
df_new = table.to_pandas()
```

Series

In Arrow, the most similar structure to a Pandas Series is an Array. It is a vector that contains data of the same type as linear memory. You can convert a Pandas Series to an Arrow Array using `pyarrow.array.from_pandas_series()`. As Arrow Arrays are always nullable, you can supply an optional mask using the `mask` parameter to mark all null-entries.

Type differences

With the current design of Pandas and Arrow, it is not possible to convert all column types unmodified. One of the main issues here is that Pandas has no support for nullable columns of arbitrary type. Also `datetime64` is currently fixed to nanosecond resolution. On the other side, Arrow might be still missing support for some types.

Pandas -> Arrow Conversion

Source Type (Pandas)	Destination Type (Arrow)
`bool`	`BOOL`
`(u)int{8,16,32,64}`	`(U)INT{8,16,32,64}`
`float32`	`FLOAT`
`float64`	`DOUBLE`
`str/unicode`	`STRING`
`pd.Categorical`	`DICTIONARY`
`pd.Timestamp`	`TIMESTAMP(unit=ns)`
`datetime.date`	`DATE`

Arrow -> Pandas Conversion

Source Type (Arrow)	Destination Type (Pandas)
`BOOL`	`bool`
`BOOL` *with nulls*	`object` (with values `True`, `False`, `None`)
`(U)INT{8,16,32,64}`	`(u)int{8,16,32,64}`
`(U)INT{8,16,32,64}` *with nulls*	`float64`
`FLOAT`	`float32`
`DOUBLE`	`float64`
`STRING`	`str`
`DICTIONARY`	`pd.Categorical`
`TIMESTAMP(unit=*)`	`pd.Timestamp(np.datetime64[ns])`
`DATE`	`pd.Timestamp(np.datetime64[ns])`

CHAPTER 4

File interfaces and Memory Maps

PyArrow features a number of file-like interfaces

Hadoop File System (HDFS)

PyArrow comes with bindings to a C++-based interface to the Hadoop File System. You connect like so:

```
import pyarrow as pa
hdfs = pa.HdfsClient(host, port, user=user, kerb_ticket=ticket_cache_path)
```

By default, `pyarrow.HdfsClient` uses libhdfs, a JNI-based interface to the Java Hadoop client. This library is loaded **at runtime** (rather than at link / library load time, since the library may not be in your LD_LIBRARY_PATH), and relies on some environment variables.

- `HADOOP_HOME`: the root of your installed Hadoop distribution. Often has *lib/native/libhdfs.so*.
- `JAVA_HOME`: the location of your Java SDK installation.
- `ARROW_LIBHDFS_DIR` (optional): explicit location of `libhdfs.so` if it is installed somewhere other than `$HADOOP_HOME/lib/native`.
- `CLASSPATH`: must contain the Hadoop jars. You can set these using:

```
export CLASSPATH=`$HADOOP_HOME/bin/hdfs classpath --glob`
```

You can also use libhdfs3, a thirdparty C++ library for HDFS from Pivotal Labs:

```
hdfs3 = pa.HdfsClient(host, port, user=user, kerb_ticket=ticket_cache_path,
                      driver='libhdfs3')
```

CHAPTER 5

Reading/Writing Parquet files

If you have built `pyarrow` with Parquet support, i.e. `parquet-cpp` was found during the build, you can read files in the Parquet format to/from Arrow memory structures. The Parquet support code is located in the `pyarrow.parquet` module and your package needs to be built with the `--with-parquet` flag for `build_ext`.

Reading Parquet

To read a Parquet file into Arrow memory, you can use the following code snippet. It will read the whole Parquet file into memory as an `Table`.

```
import pyarrow.parquet as pq

table = pq.read_table('<filename>')
```

As DataFrames stored as Parquet are often stored in multiple files, a convenience method `read_multiple_files()` is provided.

If you already have the Parquet available in memory or get it via non-file source, you can utilize `pyarrow.io.BufferReader` to read it from memory. As input to the `BufferReader` you can either supply a Python `bytes` object or a `pyarrow.io.Buffer`.

```
import pyarrow.io as paio
import pyarrow.parquet as pq

buf = ... # either bytes or paio.Buffer
reader = paio.BufferReader(buf)
table = pq.read_table(reader)
```

Writing Parquet

Given an instance of `pyarrow.table.Table`, the most simple way to persist it to Parquet is by using the *`pyarrow.parquet.write_table()`* method.

```
import pyarrow as pa
import pyarrow.parquet as pq

table = pa.Table(..)
pq.write_table(table, '<filename>')
```

By default this will write the Table as a single RowGroup using `DICTIONARY` encoding. To increase the potential of parallelism a query engine can process a Parquet file, set the `chunk_size` to a fraction of the total number of rows.

If you also want to compress the columns, you can select a compression method using the `compression` argument. Typically, `GZIP` is the choice if you want to minimize size and `SNAPPY` for performance.

Instead of writing to a file, you can also write to Python `bytes` by utilizing an `pyarrow.io.InMemoryOutputStream()`:

```
import pyarrow.io as paio
import pyarrow.parquet as pq

table = ...
output = paio.InMemoryOutputStream()
pq.write_table(table, output)
pybytes = output.get_result().to_pybytes()
```

CHAPTER 6

API Reference

Type and Schema Factory Functions

null()	
*bool_*()	
int8()	
int16()	
int32()	
int64()	
uint8()	
uint16()	
uint32()	
uint64()	
float16()	
float32()	
float64()	
time32(unit_str)	
time64(unit_str)	
timestamp(unit_str[, tz])	
date32()	
date64()	
binary(int length=-1)	Binary (PyBytes-like) type
string()	UTF8 string
decimal((int precision, int scale=0) -> DataType)	
*list_*(DataType value_type)	
struct(fields)	
dictionary(DataType index_type, Array dictionary)	Dictionary (categorical, or simply encoded) type
field(name, DataType type, ...)	Create a pyarrow.Field instance
schema(fields)	Construct pyarrow.Schema from collection of fields
from_numpy_dtype(dtype)	Convert NumPy dtype to pyarrow.DataType

pyarrow.null

pyarrow.**null**()

pyarrow.bool

pyarrow.**bool_**()

pyarrow.int8

pyarrow.**int8**()

pyarrow.int16

pyarrow.**int16**()

pyarrow.int32

pyarrow.**int32**()

pyarrow.int64

pyarrow.**int64**()

pyarrow.uint8

pyarrow.**uint8**()

pyarrow.uint16

pyarrow.**uint16**()

pyarrow.uint32

pyarrow.**uint32**()

pyarrow.uint64

pyarrow.**uint64**()

pyarrow.float16

pyarrow.**float16**()

pyarrow.float32

pyarrow.**float32**()

pyarrow.float64

pyarrow.**float64**()

pyarrow.time32

pyarrow.**time32**(*unit_str*)

pyarrow.time64

pyarrow.**time64**(*unit_str*)

pyarrow.timestamp

pyarrow.**timestamp**(*unit_str*, *tz=None*)

pyarrow.date32

pyarrow.**date32**()

pyarrow.date64

pyarrow.**date64**()

pyarrow.binary

pyarrow.**binary**(*int length=-1*)
Binary (PyBytes-like) type

Parameters **length** (*int, optional, default -1*) – If length == -1 then return a variable length binary type. If length is greater than or equal to 0 then return a fixed size binary type of width *length*.

pyarrow.string

pyarrow.**string**()
UTF8 string

pyarrow.decimal

pyarrow.**decimal**(*int precision*, *int scale=0*) → DataType

pyarrow.list

pyarrow.**list_**(*DataType value_type*)

pyarrow.struct

pyarrow.**struct**(*fields*)

pyarrow.dictionary

pyarrow.**dictionary**(*DataType index_type*, *Array dictionary*)
Dictionary (categorical, or simply encoded) type

pyarrow.field

pyarrow.**field**(*name*, *DataType type*, *bool nullable=True*, *dict metadata=None*)
Create a pyarrow.Field instance

Parameters

- **name** (*string or bytes*) –
- **type** (pyarrow.DataType) –
- **nullable** (*boolean, default True*) –
- **metadata** (*dict, default None*) – Keys and values must be coercible to bytes

Returns **field** (*pyarrow.Field*)

pyarrow.schema

pyarrow.**schema**(*fields*)
Construct pyarrow.Schema from collection of fields

Parameters **field** (*list or iterable*) –

Returns **schema** (*pyarrow.Schema*)

pyarrow.from_numpy_dtype

pyarrow.**from_numpy_dtype**(*dtype*)
Convert NumPy dtype to pyarrow.DataType

Scalar Value Types

NA
NAType
Scalar
ArrayValue
BooleanValue
Int8Value
Int16Value
Int32Value
Int64Value

Continued on next page

Table 6.2 – continued from previous page

UInt8Value
UInt16Value
UInt32Value
UInt64Value
FloatValue
DoubleValue
ListValue
BinaryValue
StringValue
FixedSizeBinaryValue
Date32Value
Date64Value
TimestampValue
DecimalValue

pyarrow.NA

pyarrow.**NA** = NA

pyarrow.NAType

class pyarrow.**NAType**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.Scalar

class pyarrow.**Scalar**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

pyarrow.ArrayValue

class pyarrow.**ArrayValue**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

pyarrow.BooleanValue

class pyarrow.**BooleanValue**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.Int8Value

class pyarrow.**Int8Value**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.Int16Value

class pyarrow.**Int16Value**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.Int32Value

class pyarrow.**Int32Value**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.Int64Value

class pyarrow.**Int64Value**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.UInt8Value

class pyarrow.**UInt8Value**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.UInt16Value

class pyarrow.**UInt16Value**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.UInt32Value

class pyarrow.**UInt32Value**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.UInt64Value

class pyarrow.**UInt64Value**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.FloatValue

class pyarrow.**FloatValue**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.DoubleValue

class pyarrow.**DoubleValue**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.ListValue

class pyarrow.**ListValue**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

Attributes

value_type

pyarrow.BinaryValue

class pyarrow.**BinaryValue**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.StringValue

class pyarrow.**StringValue**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.FixedSizeBinaryValue

class pyarrow.**FixedSizeBinaryValue**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.Date32Value

class pyarrow.**Date32Value**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.Date64Value

class pyarrow.**Date64Value**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.TimestampValue

class pyarrow.**TimestampValue**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)

pyarrow.DecimalValue

class pyarrow.**DecimalValue**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

as_py(self)	

Array Types and Constructors

array(sequence, DataType type=None, ...)	Create pyarrow.Array instance from a Python sequence
Array	
BooleanArray	
DictionaryArray	
FloatingPointArray	
IntegerArray	
Int8Array	
Int16Array	
Int32Array	
Int64Array	
NullArray	
NumericArray	
UInt8Array	
UInt16Array	
UInt32Array	
UInt64Array	
BinaryArray	
FixedSizeBinaryArray	
StringArray	
Time32Array	
Time64Array	
Date32Array	
Date64Array	
TimestampArray	
DecimalArray	
ListArray	

pyarrow.array

pyarrow.**array**(*sequence*, *DataType type=None*, *MemoryPool memory_pool=None*)

Create pyarrow.Array instance from a Python sequence

Parameters

- **sequence** (*sequence-like object of Python objects*) –
- **type** (*pyarrow.DataType, optional*) – If not passed, will be inferred from the data
- **memory_pool** (*pyarrow.MemoryPool, optional*) – If not passed, will allocate memory from the currently-set default memory pool

Returns **array** (*pyarrow.Array*)

pyarrow.Array

class pyarrow.**Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.BooleanArray

class pyarrow.**BooleanArray**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.DictionaryArray

class pyarrow.**DictionaryArray**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_arrays(indices, dictionary[, mask])	Construct Arrow DictionaryArray from array of indices (must be
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

dictionary
indices
null_count
type

pyarrow.FloatingPointArray

class pyarrow.**FloatingPointArray**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.IntegerArray

class pyarrow.**IntegerArray**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.Int8Array

class pyarrow.**Int8Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.Int16Array

class pyarrow.**Int16Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.Int32Array

class pyarrow.**Int32Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.Int64Array

class pyarrow.**Int64Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.NullArray

class pyarrow.**NullArray**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.NumericArray

class pyarrow.**NumericArray**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.UInt8Array

class pyarrow.**UInt8Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.UInt16Array

class pyarrow.**UInt16Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.UInt32Array

class pyarrow.**UInt32Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.UInt64Array

class pyarrow.**UInt64Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.BinaryArray

class pyarrow.**BinaryArray**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.FixedSizeBinaryArray

class pyarrow.**FixedSizeBinaryArray**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.StringArray

class pyarrow.**StringArray**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.Time32Array

class pyarrow.**Time32Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.Time64Array

class pyarrow.**Time64Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.Date32Array

class pyarrow.**Date32Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.Date64Array

class pyarrow.**Date64Array**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.TimestampArray

class pyarrow.**TimestampArray**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.DecimalArray

class pyarrow.**DecimalArray**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

pyarrow.ListArray

class pyarrow.**ListArray**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Array other)	
from_pandas(obj[, mask, timestamps_to_ms])	Convert pandas.Series to an Arrow Array.
isnull(self)	
slice(self[, offset, length])	Compute zero-copy slice of this array
to_pandas(self)	Convert to an array object suitable for use in pandas
to_pylist(self)	Convert to an list of native Python objects.

Attributes

null_count
type

Tables and Record Batches

ChunkedArray	Array backed via one or more memory chunks.
Column	Named vector of elements of equal type.
RecordBatch	Batch of rows of columns of equal length
Table	A collection of top-level named, equal length Arrow arrays.
get_record_batch_size(RecordBatch batch)	Return total size of serialized RecordBatch including metadata and padding

pyarrow.ChunkedArray

class pyarrow.**ChunkedArray**
Array backed via one or more memory chunks.

> **Warning:** Do not call this class's constructor directly.

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

chunk(self, i)	Select a chunk by its index
iterchunks(self)	
length(self)	

Continued on next page

Table 6.76 – continued from previous page

to_pylist(self)	Convert to a list of native Python objects.

Attributes

null_count	Number of null entires
num_chunks	Number of underlying chunks

pyarrow.Column

class pyarrow.**Column**

Named vector of elements of equal type.

> **Warning:** Do not call this class's constructor directly.

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Column other)	Check if contents of two columns are equal
from_array(field_or_name, Array arr)	
length(self)	
to_pandas(self)	Convert the arrow::Column to a pandas.Series
to_pylist(self)	Convert to a list of native Python objects.

Attributes

data	The underlying data
name	Label of the column
null_count	Number of null entires
shape	Dimensions of this columns
type	Type information for this column

pyarrow.RecordBatch

class pyarrow.**RecordBatch**

Batch of rows of columns of equal length

> **Warning:** Do not call this class's constructor directly, use one of the from_* methods instead.

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, RecordBatch other)	
from_arrays(list arrays, list names, ...)	Construct a RecordBatch from multiple pyarrow.Arrays
from_pandas(type cls, df[, schema])	Convert pandas.DataFrame to an Arrow RecordBatch
slice(self[, offset, length])	Compute zero-copy slice of this RecordBatch
to_pandas(self[, nthreads])	Convert the arrow::RecordBatch to a pandas DataFrame
to_pydict(self)	Converted the arrow::RecordBatch to an OrderedDict

Attributes

num_columns	Number of columns
num_rows	Number of rows
schema	Schema of the RecordBatch and its columns

pyarrow.Table

class pyarrow.**Table**

A collection of top-level named, equal length Arrow arrays.

> **Warning:** Do not call this class's constructor directly, use one of the from_* methods instead.

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

add_column(self, int i, Column column)	Add column to Table at position.
append_column(self, Column column)	Append column at end of columns.
column(self, index)	Select a column by its numeric index.
equals(self, Table other)	Check if contents of two tables are equal
from_arrays(arrays[, names])	Construct a Table from Arrow arrays or columns
from_batches(batches)	Construct a Table from a list of Arrow RecordBatches
from_pandas(type cls, df[, ...])	Convert pandas.DataFrame to an Arrow Table
itercolumns(self)	Iterator over all columns in their numerical order
remove_column(self, int i)	Create new Table with the indicated column removed
to_pandas(self[, nthreads])	Convert the arrow::Table to a pandas DataFrame
to_pydict(self)	Converted the arrow::Table to an OrderedDict

Attributes

num_columns	Number of columns in this table
num_rows	Number of rows in this table.
schema	Schema of the table and its columns
shape	Dimensions of the table: (#rows, #columns)

pyarrow.get_record_batch_size

pyarrow.**get_record_batch_size**(*RecordBatch batch*)
Return total size of serialized RecordBatch including metadata and padding

Tensor type and Functions

Tensor	
write_tensor(Tensor tensor, NativeFile dest)	Write pyarrow.Tensor to pyarrow.NativeFile object its current position
get_tensor_size(Tensor tensor)	Return total size of serialized Tensor including metadata and padding
read_tensor(NativeFile source)	Read pyarrow.Tensor from pyarrow.NativeFile object from current position.

pyarrow.Tensor

class pyarrow.**Tensor**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

equals(self, Tensor other)	Return true if the tensors contains exactly equal data
from_numpy(obj)	
to_numpy(self)	Convert arrow::Tensor to numpy.ndarray with zero copy

Attributes

is_contiguous
is_mutable
ndim
shape
size
strides
type

pyarrow.write_tensor

pyarrow.**write_tensor**(*Tensor tensor*, *NativeFile dest*)
Write pyarrow.Tensor to pyarrow.NativeFile object its current position

Parameters

- **tensor** (pyarrow.Tensor) –
- **dest** (pyarrow.NativeFile) –

Returns **bytes_written** (*int*) – Total number of bytes written to the file

pyarrow.get_tensor_size

pyarrow.**get_tensor_size**(*Tensor tensor*)
Return total size of serialized Tensor including metadata and padding

pyarrow.read_tensor

pyarrow.**read_tensor**(*NativeFile source*)
Read pyarrow.Tensor from pyarrow.NativeFile object from current position. If the file source supports zero copy (e.g. a memory map), then this operation does not allocate any memory

Parameters **source** (pyarrow.NativeFile) –

Returns **tensor** (*Tensor*)

Input / Output and Shared Memory

Buffer	
BufferReader	Zero-copy reader from objects convertible to Arrow buffer
InMemoryOutputStream	
NativeFile	
MemoryMappedFile	Supports 'r', 'r+w', 'w' modes
memory_map(path[, mode])	Open memory map at file path.
create_memory_map(path, size)	Create memory map at indicated path of the given size, return open
PythonFile	

pyarrow.Buffer

class pyarrow.**Buffer**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

to_pybytes(self)

Attributes

parent
size

pyarrow.BufferReader

class pyarrow.**BufferReader**
Zero-copy reader from objects convertible to Arrow buffer

Parameters **obj** (*Python bytes or pyarrow.io.Buffer*) –

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

close(self)	
download(self, stream_or_path[, buffer_size])	Read file completely to local path (rather than reading completely into memory).
read(self[, nbytes])	
read_buffer(self[, nbytes])	
seek(self, int64_t position)	
size(self)	
tell(self)	
upload(self, stream[, buffer_size])	Pipe file-like object to file
write(self, data)	Write byte from any object implementing buffer protocol (bytes,

pyarrow.InMemoryOutputStream

class pyarrow.**InMemoryOutputStream**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

close(self)	
download(self, stream_or_path[, buffer_size])	Read file completely to local path (rather than reading completely into memory).
get_result(self)	
read(self[, nbytes])	
read_buffer(self[, nbytes])	
seek(self, int64_t position)	
size(self)	
tell(self)	
upload(self, stream[, buffer_size])	Pipe file-like object to file
write(self, data)	Write byte from any object implementing buffer protocol (bytes,

pyarrow.NativeFile

class pyarrow.**NativeFile**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

close(self)	
download(self, stream_or_path[, buffer_size])	Read file completely to local path (rather than reading completely into memory).
read(self[, nbytes])	
read_buffer(self[, nbytes])	
seek(self, int64_t position)	
size(self)	
tell(self)	
upload(self, stream[, buffer_size])	Pipe file-like object to file
write(self, data)	Write byte from any object implementing buffer protocol (bytes,

pyarrow.MemoryMappedFile

class pyarrow.**MemoryMappedFile**

Supports 'r', 'r+w', 'w' modes

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

close(self)	
create(path, size)	
download(self, stream_or_path[, buffer_size])	Read file completely to local path (rather than reading completely into memory).
open(self, path[, mode])	
read(self[, nbytes])	
read_buffer(self[, nbytes])	
seek(self, int64_t position)	
size(self)	
tell(self)	
upload(self, stream[, buffer_size])	Pipe file-like object to file
write(self, data)	Write byte from any object implementing buffer protocol (bytes,

pyarrow.memory_map

pyarrow.**memory_map**(*path*, *mode='r'*)

Open memory map at file path. Size of the memory map cannot change

Parameters

- **path** (string) –
- **mode** ({'r', 'w'}, default 'r') –

Returns **mmap** (*MemoryMappedFile*)

pyarrow.create_memory_map

pyarrow.**create_memory_map**(*path*, *size*)

Create memory map at indicated path of the given size, return open writeable file object

Parameters

- **path** (string) –
- **size** (*int*) –

Returns **mmap** (*MemoryMappedFile*)

pyarrow.PythonFile

class pyarrow.**PythonFile**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

close(self)	
download(self, stream_or_path[, buffer_size])	Read file completely to local path (rather than reading completely into memory).
read(self[, nbytes])	
read_buffer(self[, nbytes])	
seek(self, int64_t position)	
size(self)	
tell(self)	
upload(self, stream[, buffer_size])	Pipe file-like object to file
write(self, data)	Write byte from any object implementing buffer protocol (bytes,

Interprocess Communication and Messaging

FileReader(source[, footer_offset])	Class for reading Arrow record batch data from the Arrow binary file format
FileWriter(sink, schema)	Writer to create the Arrow binary file format
StreamReader(source)	Reader for the Arrow streaming binary format
StreamWriter(sink, schema)	Writer for the Arrow streaming binary format

pyarrow.FileReader

class pyarrow.**FileReader**(*source*, *footer_offset=None*)
Class for reading Arrow record batch data from the Arrow binary file format

Parameters

- **source** (*str, pyarrow.NativeFile, or file-like Python object*) – Either a file path, or a readable file object
- **footer_offset** (*int, default None*) – If the file is embedded in some larger file, this is the byte offset to the very end of the file data

__init__(*source*, *footer_offset=None*)

Methods

__init__(source[, footer_offset])	
get_batch(self, int i)	
get_record_batch	_FileReader.get_batch(self, int i)
read_all(self)	Read all record batches as a pyarrow.Table

Attributes

num_record_batches

pyarrow.FileWriter

class pyarrow.**FileWriter**(*sink*, *schema*)

Writer to create the Arrow binary file format

Parameters

- **sink** (*str, pyarrow.NativeFile, or file-like Python object*) – Either a file path, or a writeable file object
- **schema** (pyarrow.Schema) – The Arrow schema for data to be written to the file

__init__(*sink*, *schema*)

Methods

__init__(sink, schema)	
close(self)	
write_batch(self, RecordBatch batch)	

pyarrow.StreamReader

class pyarrow.**StreamReader**(*source*)

Reader for the Arrow streaming binary format

Parameters **source** (*str, pyarrow.NativeFile, or file-like Python object*) – Either a file path, or a readable file object

__init__(*source*)

Methods

__init__(source)	
get_next_batch(self)	Read next RecordBatch from the stream.
read_all(self)	Read all record batches as a pyarrow.Table

Attributes

schema

pyarrow.StreamWriter

class pyarrow.**StreamWriter**(*sink*, *schema*)

Writer for the Arrow streaming binary format

Parameters

- **sink** (*str, pyarrow.NativeFile, or file-like Python object*) – Either a file path, or a writeable file object
- **schema** (pyarrow.Schema) – The Arrow schema for data to be written to the file

__init__(*sink*, *schema*)

Methods

__init__(sink, schema)
close(self)
write_batch(self, RecordBatch batch)

Memory Pools

MemoryPool	
default_memory_pool()	
jemalloc_memory_pool()	Returns a jemalloc-based memory allocator, which can be passed to
total_allocated_bytes()	
set_memory_pool(MemoryPool pool)	

pyarrow.MemoryPool

class pyarrow.**MemoryPool**

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

bytes_allocated(self)

pyarrow.default_memory_pool

pyarrow.**default_memory_pool**()

pyarrow.jemalloc_memory_pool

pyarrow.**jemalloc_memory_pool**()
Returns a jemalloc-based memory allocator, which can be passed to pyarrow.set_memory_pool

pyarrow.total_allocated_bytes

pyarrow.**total_allocated_bytes**()

pyarrow.set_memory_pool

pyarrow.**set_memory_pool**(*MemoryPool pool*)

Type Classes

DataType	
DecimalType	
DictionaryType	
FixedSizeBinaryType	
Time32Type	
Time64Type	
TimestampType	
Field	Represents a named field, with a data type, nullability, and optional
Schema	

pyarrow.DataType

class pyarrow.**DataType**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

to_pandas_dtype(self)	Return the NumPy dtype that would be used for storing this

pyarrow.DecimalType

class pyarrow.**DecimalType**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

to_pandas_dtype(self)	Return the NumPy dtype that would be used for storing this

Attributes

byte_width

pyarrow.DictionaryType

class pyarrow.**DictionaryType**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

to_pandas_dtype(self)	Return the NumPy dtype that would be used for storing this

pyarrow.FixedSizeBinaryType

class pyarrow.**FixedSizeBinaryType**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

to_pandas_dtype(self)	Return the NumPy dtype that would be used for storing this

Attributes

byte_width

pyarrow.Time32Type

class pyarrow.**Time32Type**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

to_pandas_dtype(self)	Return the NumPy dtype that would be used for storing this

Attributes

unit

pyarrow.Time64Type

class pyarrow.**Time64Type**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

to_pandas_dtype(self)	Return the NumPy dtype that would be used for storing this

Attributes

unit

pyarrow.TimestampType

class pyarrow.**TimestampType**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

to_pandas_dtype(self)	Return the NumPy dtype that would be used for storing this

Attributes

tz
unit

pyarrow.Field

class pyarrow.**Field**
Represents a named field, with a data type, nullability, and optional metadata

Notes

Do not use this class's constructor directly; use pyarrow.field

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

add_metadata(self, dict metadata)	Add metadata as dict of string keys and values to Field
equals(self, Field other)	Test if this field is equal to the other
remove_metadata(self)	Create new field without metadata, if any

Attributes

metadata
name
nullable
type

pyarrow.Schema

class pyarrow.**Schema**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

add_metadata(self, dict metadata)	Add metadata as dict of string keys and values to Schema
equals(self, other)	Test if this schema is equal to the other
field_by_name(self, name)	Access a field by its name rather than the column index.
remove_metadata(self)	Create new schema without metadata, if any

Attributes

metadata
names

Apache Parquet

ParquetDataset(path_or_paths[, filesystem, ...])	Encapsulates details of reading a complete Parquet dataset possibly
ParquetFile(source[, metadata])	Reader interface for a single Parquet file

Continued on next page

Table 6.121 – continued from previous page

read_table(source[, columns, nthreads, metadata])	Read a Table from Parquet format
write_metadata(schema, where[, version])	Write metadata-only Parquet file from schema
write_table(table, where[, row_group_size, ...])	Write a Table to Parquet format

pyarrow.parquet.ParquetDataset

class pyarrow.parquet.**ParquetDataset**(*path_or_paths*, *filesystem=None*, *schema=None*, *metadata=None*, *split_row_groups=False*, *validate_schema=True*)

Encapsulates details of reading a complete Parquet dataset possibly consisting of multiple files and partitions in subdirectories

Parameters

- **path_or_paths** (*str or List[str]*) – A directory name, single file name, or list of file names
- **filesystem** (*Filesystem, default None*) – If nothing passed, paths assumed to be found in the local on-disk filesystem
- **metadata** (*pyarrow.parquet.FileMetaData*) – Use metadata obtained elsewhere to validate file schemas
- **schema** (*pyarrow.parquet.Schema*) – Use schema obtained elsewhere to validate file schemas. Alternative to metadata parameter
- **split_row_groups** (*boolean, default False*) – Divide files into pieces for each row group in the file
- **validate_schema** (*boolean, default True*) – Check that individual file schemas are all the same / compatible

__init__(*path_or_paths*, *filesystem=None*, *schema=None*, *metadata=None*, *split_row_groups=False*, *validate_schema=True*)

Methods

__init__(path_or_paths[, filesystem, ...])	
read([columns, nthreads])	Read multiple Parquet files as a single pyarrow.Table
validate_schemas()	

pyarrow.parquet.ParquetFile

class pyarrow.parquet.**ParquetFile**(*source*, *metadata=None*)

Reader interface for a single Parquet file

Parameters

- **source** (*str or pyarrow.io.NativeFile*) – Readable source. For passing Python file objects or byte buffers, see pyarrow.io.PythonFileInterface or pyarrow.io.BufferReader.
- **metadata** (*ParquetFileMetadata, default None*) – Use existing metadata object, rather than reading from file.

__init__(*source*, *metadata=None*)

Methods

`__init__`(source[, metadata])	
`read`([columns, nthreads])	Read a Table from Parquet format
`read_row_group`(i[, columns, nthreads])	Read a single row group from a Parquet file

Attributes

`metadata`
`num_row_groups`
`schema`

pyarrow.parquet.read_table

pyarrow.parquet.**read_table**(*source*, *columns=None*, *nthreads=1*, *metadata=None*)

Read a Table from Parquet format

Parameters

- **source** (*str or pyarrow.io.NativeFile*) – Location of Parquet dataset. If a string passed, can be a single file name or directory name. For passing Python file objects or byte buffers, see pyarrow.io.PythonFileInterface or pyarrow.io.BufferReader.
- **columns** (*list*) – If not None, only these columns will be read from the file.
- **nthreads** (*int, default 1*) – Number of columns to read in parallel. Requires that the underlying file source is threadsafe
- **metadata** (*FileMetaData*) – If separately computed

Returns *pyarrow.Table* – Content of the file as a table (of columns)

pyarrow.parquet.write_metadata

pyarrow.parquet.**write_metadata**(*schema*, *where*, *version='1.0'*)

Write metadata-only Parquet file from schema

Parameters

- **schema** (pyarrow.Schema) –
- **where** (*string or pyarrow.io.NativeFile*) –
- **version** (*{"1.0", "2.0"}, default "1.0"*) – The Parquet format version, defaults to 1.0

pyarrow.parquet.write_table

pyarrow.parquet.**write_table**(*table*, *where*, *row_group_size=None*, *version='1.0'*, *use_dictionary=True*, *compression='snappy'*, ***kwargs*)

Write a Table to Parquet format

Parameters

- **table** (pyarrow.Table) –

- **where** (*string or pyarrow.io.NativeFile*) –
- **row_group_size** (*int, default None*) – The maximum number of rows in each Parquet RowGroup. As a default, we will write a single RowGroup per file.
- **version** (*{"1.0", "2.0"}, default "1.0"*) – The Parquet format version, defaults to 1.0
- **use_dictionary** (*bool or list*) – Specify if we should use dictionary encoding in general or only for some columns.
- **compression** (*str or dict*) – Specify the compression codec, either on a general basis or per-column.

CHAPTER 7

Getting Involved

Right now the primary audience for Apache Arrow are the developers of data systems; most people will use Apache Arrow indirectly through systems that use it for internal data handling and interoperating with other Arrow-enabled systems.

Even if you do not plan to contribute to Apache Arrow itself or Arrow integrations in other projects, we'd be happy to have you involved:

- Join the mailing list: send an email to dev-subscribe@arrow.apache.org. Share your ideas and use cases for the project or read through the Archive.
- Follow our activity on JIRA
- Learn the Format / Specification
- Chat with us on Slack

CHAPTER 8

jemalloc MemoryPool

Arrow's default `MemoryPool` uses the system's allocator through the POSIX APIs. Although this already provides aligned allocation, the POSIX interface doesn't support aligned reallocation. The default reallocation strategy is to allocate a new region, copy over the old data and free the previous region. Using jemalloc we can simply extend the existing memory allocation to the requested size. While this may still be linear in the size of allocated memory, it is magnitudes faster as only the page mapping in the kernel is touched, not the actual data.

The `jemalloc` allocator is not enabled by default to allow the use of the system allocator and/or other allocators like `tcmalloc`. You can either explicitly make it the default allocator or pass it only to single operations.

```
import pyarrow as pa

jemalloc_pool = pyarrow.jemalloc_memory_pool()

# Explicitly use jemalloc for allocating memory for an Arrow Table object
array = pa.Array.from_pylist([1, 2, 3], memory_pool=jemalloc_pool)

# Set the global pool
pyarrow.set_memory_pool(jemalloc_pool)
# This operation has no explicit MemoryPool specified and will thus will
# also use jemalloc for its allocations.
array = pa.Array.from_pylist([1, 2, 3])
```

Symbols

A

T

U

W