// Licensed to the Apache Software Foundation (ASF) under one

// or more contributor license agreements. See the NOTICE file

// distributed with this work for additional information

// regarding copyright ownership. The ASF licenses this file

// to you under the Apache License, Version 2.0 (the

// "License"); you may not use this file except in compliance

// with the License. You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing,

// software distributed under the License is distributed on an

// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY

// KIND, either express or implied. See the License for the

// specific language governing permissions and limitations

// under the License.

#include <memory>

#include <sstream>

#include <utility>

#include "arrow/python/extension\_type.h"

#include "arrow/python/helpers.h"

#include "arrow/python/pyarrow.h"

#include "arrow/util/checked\_cast.h"

#include "arrow/util/logging.h"

namespace arrow {

using internal::checked\_cast;

namespace py {

namespace {

// Serialize a Python ExtensionType instance

Status SerializeExtInstance(PyObject\* type\_instance, std::string\* out) {

OwnedRef res(

cpp\_PyObject\_CallMethod(type\_instance, "\_\_arrow\_ext\_serialize\_\_", nullptr));

if (!res) {

return ConvertPyError();

}

if (!PyBytes\_Check(res.obj())) {

return Status::TypeError(

"\_\_arrow\_ext\_serialize\_\_ should return bytes object, "

"got ",

internal::PyObject\_StdStringRepr(res.obj()));

}

\*out = internal::PyBytes\_AsStdString(res.obj());

return Status::OK();

}

// Deserialize a Python ExtensionType instance

PyObject\* DeserializeExtInstance(PyObject\* type\_class,

std::shared\_ptr<DataType> storage\_type,

const std::string& serialized\_data) {

OwnedRef storage\_ref(wrap\_data\_type(storage\_type));

if (!storage\_ref) {

return nullptr;

}

OwnedRef data\_ref(PyBytes\_FromStringAndSize(

serialized\_data.data(), static\_cast<Py\_ssize\_t>(serialized\_data.size())));

if (!data\_ref) {

return nullptr;

}

return cpp\_PyObject\_CallMethod(type\_class, "\_\_arrow\_ext\_deserialize\_\_", "OO",

storage\_ref.obj(), data\_ref.obj());

}

} // namespace

static const char\* kExtensionName = "arrow.py\_extension\_type";

std::string PyExtensionType::ToString(bool show\_metadata) const {

PyAcquireGIL lock;

std::stringstream ss;

OwnedRef instance(GetInstance());

ss << "extension<" << this->extension\_name() << "<" << Py\_TYPE(instance.obj())->tp\_name

<< ">>";

return ss.str();

}

PyExtensionType::PyExtensionType(std::shared\_ptr<DataType> storage\_type, PyObject\* typ,

PyObject\* inst)

: ExtensionType(storage\_type),

extension\_name\_(kExtensionName),

type\_class\_(typ),

type\_instance\_(inst) {}

PyExtensionType::PyExtensionType(std::shared\_ptr<DataType> storage\_type,

std::string extension\_name, PyObject\* typ,

PyObject\* inst)

: ExtensionType(storage\_type),

extension\_name\_(std::move(extension\_name)),

type\_class\_(typ),

type\_instance\_(inst) {}

bool PyExtensionType::ExtensionEquals(const ExtensionType& other) const {

PyAcquireGIL lock;

if (other.extension\_name() != extension\_name()) {

return false;

}

const auto& other\_ext = checked\_cast<const PyExtensionType&>(other);

int res = -1;

if (!type\_instance\_) {

if (other\_ext.type\_instance\_) {

return false;

}

// Compare Python types

res = PyObject\_RichCompareBool(type\_class\_.obj(), other\_ext.type\_class\_.obj(), Py\_EQ);

} else {

if (!other\_ext.type\_instance\_) {

return false;

}

// Compare Python instances

OwnedRef left(GetInstance());

OwnedRef right(other\_ext.GetInstance());

if (!left || !right) {

goto error;

}

res = PyObject\_RichCompareBool(left.obj(), right.obj(), Py\_EQ);

}

if (res == -1) {

goto error;

}

return res == 1;

error:

// Cannot propagate error

PyErr\_WriteUnraisable(nullptr);

return false;

}

std::shared\_ptr<Array> PyExtensionType::MakeArray(std::shared\_ptr<ArrayData> data) const {

DCHECK\_EQ(data->type->id(), Type::EXTENSION);

return std::make\_shared<ExtensionArray>(data);

}

std::string PyExtensionType::Serialize() const {

DCHECK(type\_instance\_);

return serialized\_;

}

Result<std::shared\_ptr<DataType>> PyExtensionType::Deserialize(

std::shared\_ptr<DataType> storage\_type, const std::string& serialized\_data) const {

PyAcquireGIL lock;

if (import\_pyarrow()) {

return ConvertPyError();

}

OwnedRef res(DeserializeExtInstance(type\_class\_.obj(), storage\_type, serialized\_data));

if (!res) {

return ConvertPyError();

}

return unwrap\_data\_type(res.obj());

}

PyObject\* PyExtensionType::GetInstance() const {

if (!type\_instance\_) {

PyErr\_SetString(PyExc\_TypeError, "Not an instance");

return nullptr;

}

DCHECK(PyWeakref\_CheckRef(type\_instance\_.obj()));

PyObject\* inst = PyWeakref\_GET\_OBJECT(type\_instance\_.obj());

if (inst != Py\_None) {

// Cached instance still alive

Py\_INCREF(inst);

return inst;

} else {

// Must reconstruct from serialized form

// XXX cache again?

return DeserializeExtInstance(type\_class\_.obj(), storage\_type\_, serialized\_);

}

}

Status PyExtensionType::SetInstance(PyObject\* inst) const {

// Check we have the right type

PyObject\* typ = reinterpret\_cast<PyObject\*>(Py\_TYPE(inst));

if (typ != type\_class\_.obj()) {

return Status::TypeError("Unexpected Python ExtensionType class ",

internal::PyObject\_StdStringRepr(typ), " expected ",

internal::PyObject\_StdStringRepr(type\_class\_.obj()));

}

PyObject\* wr = PyWeakref\_NewRef(inst, nullptr);

if (wr == NULL) {

return ConvertPyError();

}

type\_instance\_.reset(wr);

return SerializeExtInstance(inst, &serialized\_);

}

Status PyExtensionType::FromClass(const std::shared\_ptr<DataType> storage\_type,

const std::string extension\_name, PyObject\* typ,

std::shared\_ptr<ExtensionType>\* out) {

Py\_INCREF(typ);

out->reset(new PyExtensionType(storage\_type, std::move(extension\_name), typ));

return Status::OK();

}

Status RegisterPyExtensionType(const std::shared\_ptr<DataType>& type) {

DCHECK\_EQ(type->id(), Type::EXTENSION);

auto ext\_type = std::dynamic\_pointer\_cast<ExtensionType>(type);

return RegisterExtensionType(ext\_type);

}

Status UnregisterPyExtensionType(const std::string& type\_name) {

return UnregisterExtensionType(type\_name);

}

std::string PyExtensionName() { return kExtensionName; }

} // namespace py

} // namespace arrow