// 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 "arrow/python/filesystem.h"

#include "arrow/util/logging.h"

namespace arrow {

using fs::FileInfo;

using fs::FileSelector;

namespace py {

namespace fs {

PyFileSystem::PyFileSystem(PyObject\* handler, PyFileSystemVtable vtable)

: handler\_(handler), vtable\_(std::move(vtable)) {

Py\_INCREF(handler);

}

PyFileSystem::~PyFileSystem() {}

std::shared\_ptr<PyFileSystem> PyFileSystem::Make(PyObject\* handler,

PyFileSystemVtable vtable) {

return std::make\_shared<PyFileSystem>(handler, std::move(vtable));

}

std::string PyFileSystem::type\_name() const {

std::string result;

auto st = SafeCallIntoPython([&]() -> Status {

vtable\_.get\_type\_name(handler\_.obj(), &result);

if (PyErr\_Occurred()) {

PyErr\_WriteUnraisable(handler\_.obj());

}

return Status::OK();

});

ARROW\_UNUSED(st);

return result;

}

bool PyFileSystem::Equals(const FileSystem& other) const {

bool result;

auto st = SafeCallIntoPython([&]() -> Status {

result = vtable\_.equals(handler\_.obj(), other);

if (PyErr\_Occurred()) {

PyErr\_WriteUnraisable(handler\_.obj());

}

return Status::OK();

});

ARROW\_UNUSED(st);

return result;

}

Result<FileInfo> PyFileSystem::GetFileInfo(const std::string& path) {

FileInfo info;

auto st = SafeCallIntoPython([&]() -> Status {

vtable\_.get\_file\_info(handler\_.obj(), path, &info);

return CheckPyError();

});

RETURN\_NOT\_OK(st);

return info;

}

Result<std::vector<FileInfo>> PyFileSystem::GetFileInfo(

const std::vector<std::string>& paths) {

std::vector<FileInfo> infos;

auto st = SafeCallIntoPython([&]() -> Status {

vtable\_.get\_file\_info\_vector(handler\_.obj(), paths, &infos);

return CheckPyError();

});

RETURN\_NOT\_OK(st);

return infos;

}

Result<std::vector<FileInfo>> PyFileSystem::GetFileInfo(const FileSelector& select) {

std::vector<FileInfo> infos;

auto st = SafeCallIntoPython([&]() -> Status {

vtable\_.get\_file\_info\_selector(handler\_.obj(), select, &infos);

return CheckPyError();

});

RETURN\_NOT\_OK(st);

return infos;

}

Status PyFileSystem::CreateDir(const std::string& path, bool recursive) {

return SafeCallIntoPython([&]() -> Status {

vtable\_.create\_dir(handler\_.obj(), path, recursive);

return CheckPyError();

});

}

Status PyFileSystem::DeleteDir(const std::string& path) {

return SafeCallIntoPython([&]() -> Status {

vtable\_.delete\_dir(handler\_.obj(), path);

return CheckPyError();

});

}

Status PyFileSystem::DeleteDirContents(const std::string& path, bool missing\_dir\_ok) {

return SafeCallIntoPython([&]() -> Status {

vtable\_.delete\_dir\_contents(handler\_.obj(), path, missing\_dir\_ok);

return CheckPyError();

});

}

Status PyFileSystem::DeleteRootDirContents() {

return SafeCallIntoPython([&]() -> Status {

vtable\_.delete\_root\_dir\_contents(handler\_.obj());

return CheckPyError();

});

}

Status PyFileSystem::DeleteFile(const std::string& path) {

return SafeCallIntoPython([&]() -> Status {

vtable\_.delete\_file(handler\_.obj(), path);

return CheckPyError();

});

}

Status PyFileSystem::Move(const std::string& src, const std::string& dest) {

return SafeCallIntoPython([&]() -> Status {

vtable\_.move(handler\_.obj(), src, dest);

return CheckPyError();

});

}

Status PyFileSystem::CopyFile(const std::string& src, const std::string& dest) {

return SafeCallIntoPython([&]() -> Status {

vtable\_.copy\_file(handler\_.obj(), src, dest);

return CheckPyError();

});

}

Result<std::shared\_ptr<io::InputStream>> PyFileSystem::OpenInputStream(

const std::string& path) {

std::shared\_ptr<io::InputStream> stream;

auto st = SafeCallIntoPython([&]() -> Status {

vtable\_.open\_input\_stream(handler\_.obj(), path, &stream);

return CheckPyError();

});

RETURN\_NOT\_OK(st);

return stream;

}

Result<std::shared\_ptr<io::RandomAccessFile>> PyFileSystem::OpenInputFile(

const std::string& path) {

std::shared\_ptr<io::RandomAccessFile> stream;

auto st = SafeCallIntoPython([&]() -> Status {

vtable\_.open\_input\_file(handler\_.obj(), path, &stream);

return CheckPyError();

});

RETURN\_NOT\_OK(st);

return stream;

}

Result<std::shared\_ptr<io::OutputStream>> PyFileSystem::OpenOutputStream(

const std::string& path, const std::shared\_ptr<const KeyValueMetadata>& metadata) {

std::shared\_ptr<io::OutputStream> stream;

auto st = SafeCallIntoPython([&]() -> Status {

vtable\_.open\_output\_stream(handler\_.obj(), path, metadata, &stream);

return CheckPyError();

});

RETURN\_NOT\_OK(st);

return stream;

}

Result<std::shared\_ptr<io::OutputStream>> PyFileSystem::OpenAppendStream(

const std::string& path, const std::shared\_ptr<const KeyValueMetadata>& metadata) {

std::shared\_ptr<io::OutputStream> stream;

auto st = SafeCallIntoPython([&]() -> Status {

vtable\_.open\_append\_stream(handler\_.obj(), path, metadata, &stream);

return CheckPyError();

});

RETURN\_NOT\_OK(st);

return stream;

}

Result<std::string> PyFileSystem::NormalizePath(std::string path) {

std::string normalized;

auto st = SafeCallIntoPython([&]() -> Status {

vtable\_.normalize\_path(handler\_.obj(), path, &normalized);

return CheckPyError();

});

RETURN\_NOT\_OK(st);

return normalized;

}

} // namespace fs

} // namespace py

} // namespace arrow