PyHDFS

Jing Wang

Jan 28, 2024

CONTENTS:

For a quick introduction, see the main README. For detailed documentation on available methods, see *pyhdfs.HdfsClient*.

CHAPTER

ONE

PYHDFS MODULE

WebHDFS client with support for NN HA and automatic error checking

For details on the WebHDFS endpoints, see the Hadoop documentation:

- https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/WebHDFS.html
- https://hadoop.apache.org/docs/current/api/org/apache/hadoop/fs/FileSystem.html
- https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/filesystem/filesystem.html

class pyhdfs.**ContentSummary**(***kwargs: object*)

Bases: _BoilerplateClass

Parameters

- **directoryCount** (*int*) – The number of directories.
- **fileCount** (*int*) – The number of files.
- **length** (*int*) – The number of bytes used by the content.
- **quota** (*int*) – The namespace quota of this directory.
- **spaceConsumed** (*int*) – The disk space consumed by the content.
- **spaceQuota** (*int*) – The disk space quota.
- **typeQuota** (*Dict[str, TypeQuota]*) – Quota usage for ARCHIVE, DISK, SSD

directoryCount: int

fileCount: int

length: int

quota: int

spaceConsumed: int

spaceQuota: int

typeQuota: Dict[str, *TypeQuota*]

class pyhdfs.**FileChecksum**(***kwargs: object*)

Bases: _BoilerplateClass

Parameters

- **algorithm** (*str*) – The name of the checksum algorithm.
- **bytes** (*str*) – The byte sequence of the checksum in hexadecimal.

- **length** (*int*) – The length of the bytes (not the length of the string).

algorithm: str

bytes: str

length: int

class pyhdfs.**FileStatus**(***kwargs: object*)

Bases: _BoilerplateClass

Parameters

- **accessTime** (*int*) – The access time.
- **blockSize** (*int*) – The block size of a file.
- **group** (*str*) – The group owner.
- **length** (*int*) – The number of bytes in a file.
- **modificationTime** (*int*) – The modification time.
- **owner** (*str*) – The user who is the owner.
- **pathSuffix** (*str*) – The path suffix.
- **permission** (*str*) – The permission represented as a octal string.
- **replication** (*int*) – The number of replication of a file.
- **symlink** (*Optional[str]*) – The link target of a symlink.
- **type** (*str*) – The type of the path object.
- **childrenNum** (*int*) – How many children this directory has, or 0 for files.

accessTime: int

blockSize: int

childrenNum: int

group: str

length: int

modificationTime: int

owner: str

pathSuffix: str

permission: str

replication: int

symlink: str | None

type: str

exception pyhdfs.**HdfsAccessControlException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsIOException*

class pyhdfs.**HdfsClient**(*hosts: str | Iterable[str] = 'localhost'*, *randomize_hosts: bool = True*, *user_name: str | None = None*, *timeout: float = 20*, *max_tries: int = 2*, *retry_delay: float = 5*, *requests_session: Session | None = None*, *requests_kwargs: Dict[str, Any] | None = None*)

Bases: `object`

HDFS client backed by WebHDFS.

All functions take arbitrary query parameters to pass to WebHDFS, in addition to any documented keyword arguments. In particular, any function will accept `user.name`, which for convenience may be passed as `user_name`.

If multiple HA NameNodes are given, all functions submit HTTP requests to both NameNodes until they find the active NameNode.

Parameters

- **hosts** (*list or str*) – List of NameNode HTTP host:port strings, either as `list` or a comma separated string. Port defaults to 50070 if left unspecified. Note that in Hadoop 3, the default NameNode HTTP port changed to 9870; the old default of 50070 is left as-is for backwards compatibility.
- **randomize_hosts** (*bool*) – By default randomize host selection.
- **user_name** – What Hadoop user to run as. Defaults to the `HADOOP_USER_NAME` environment variable if present, otherwise `getpass.getuser()`.
- **timeout** (*float*) – How long to wait on a single NameNode in seconds before moving on. In some cases the standby NameNode can be unresponsive (e.g. loading fsimage or checkpointing), so we don't want to block on it.
- **max_tries** (*int*) – How many times to retry a request for each NameNode. If NN1 is standby and NN2 is active, we might first contact NN1 and then observe a failover to NN1 when we contact NN2. In this situation we want to retry against NN1.
- **retry_delay** (*float*) – How long to wait in seconds before going through NameNodes again
- **requests_session** – A `requests.Session` object for advanced usage. If absent, this class will use the default requests behavior of making a new session per HTTP request. Caller is responsible for closing session.
- **requests_kwargs** – Additional `**kwargs` to pass to requests

append(*path: str*, *data: bytes | IO[bytes]*, ***kwargs: str | int | None | List[str]*) → None

Append to the given file.

Parameters

- **data** – `bytes` or a `file`-like object
- **buffersize** (*int*) – The size of the buffer used in transferring data.

concat(*target: str*, *sources: List[str]*, ***kwargs: str | int | None | List[str]*) → None

Concat existing files together.

For preconditions, see https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/filesystem/filesystem.html#void_concatPath_p_Path_sources

Parameters

- **target** – the path to the target destination.
- **sources** (*list*) – the paths to the sources to use for the concatenation.

copy_from_local(*localsrc: str*, *dest: str*, ***kwargs: str | int | None | List[str]*) → None

Copy a single file from the local file system to `dest`

Takes all arguments that *create()* takes.

copy_to_local(*src: str*, *localdest: str*, ***kwargs: str | int | None | List[str]*) → None

Copy a single file from `src` to the local file system

Takes all arguments that *open()* takes.

create(*path: str*, *data: IO[bytes] | bytes*, ***kwargs: str | int | None | List[str]*) → None

Create a file at the given path.

Parameters

- **data** – `bytes` or a `file`-like object to upload
- **overwrite** (*bool*) – If a file already exists, should it be overwritten?
- **blocksize** (*long*) – The block size of a file.
- **replication** (*short*) – The number of replications of a file.
- **permission** (*octal*) – The permission of a file/directory. Any radix-8 integer (leading zeros may be omitted.)
- **buffersize** (*int*) – The size of the buffer used in transferring data.

create_snapshot(*path: str*, ***kwargs: str | int | None | List[str]*) → str

Create a snapshot

Parameters

- **path** – The directory where snapshots will be taken
- **snapshotname** – The name of the snapshot

Returns

the snapshot path

create_symlink(*link: str*, *destination: str*, ***kwargs: str | int | None | List[str]*) → None

Create a symbolic link at `link` pointing to `destination`.

Parameters

- **link** – the path to be created that points to target
- **destination** – the target of the symbolic link
- **createParent** (*bool*) – If the parent directories do not exist, should they be created?

Raises

HdfsUnsupportedOperationException – This feature doesn't actually work, at least on CDH 5.3.0.

delete(*path: str*, ***kwargs: str | int | None | List[str]*) → bool

Delete a file.

Parameters

recursive (*bool*) – If path is a directory and set to true, the directory is deleted else throws an exception. In case of a file the recursive can be set to either true or false.

Returns

true if delete is successful else false.

Return type
bool

delete_snapshot(*path: str*, *snapshotname: str*, ***kwargs: str | int | None | List[str]*) → None

Delete a snapshot of a directory

exists(*path: str*, ***kwargs: str | int | None | List[str]*) → bool

Return true if the given path exists

get_active_namenode(*max_staleness: float | None = None*) → str

Return the address of the currently active NameNode.

Parameters
max_staleness (*float*) – This function caches the active NameNode. If this age of this cached result is less than `max_staleness` seconds, return it. Otherwise, or if this parameter is None, do a lookup.

Raises
HdfsNoServerException – can't find an active NameNode

get_content_summary(*path: str*, ***kwargs: str | int | None | List[str]*) → *ContentSummary*

Return the *ContentSummary* of a given Path.

get_file_checksum(*path: str*, ***kwargs: str | int | None | List[str]*) → *FileChecksum*

Get the checksum of a file.

Return type
FileChecksum

get_file_status(*path: str*, ***kwargs: str | int | None | List[str]*) → *FileStatus*

Return a *FileStatus* object that represents the path.

get_home_directory(***kwargs: str | int | None | List[str]*) → str

Return the current user's home directory in this filesystem.

get_xattrs(*path: str*, *xattr_name: str | List[str] | None = None*, *encoding: str = 'text'*, ***kwargs: str | int | None | List[str]*) → Dict[str, bytes | str | None]

Get one or more xattr values for a file or directory.

Parameters

- **xattr_name** – `str` to get one attribute, `list` to get multiple attributes, `None` to get all attributes.
- **encoding** – `text` | `hex` | `base64`, defaults to `text`

Returns
Dictionary mapping xattr name to value. With text encoding, the value will be a unicode string. With hex or base64 encoding, the value will be a byte array.

Return type
dict

list_status(*path: str*, ***kwargs: str | int | None | List[str]*) → List[*FileStatus*]

List the statuses of the files/directories in the given path if the path is a directory.

Return type
`list` of *FileStatus* objects

list_xattrs(*path: str*, ***kwargs: str | int | None | List[str]*) → List[str]

Get all of the xattr names for a file or directory.

Return type
list

listdir(*path: str*, ***kwargs: str | int | None | List[str]*) → List[str]

Return a list containing names of files in the given path

mkdirs(*path: str*, ***kwargs: str | int | None | List[str]*) → bool

Create a directory with the provided permission.

The permission of the directory is set to be the provided permission as in setPermission, not permission&~umask.

Parameters
permission (*octal*) – The permission of a file/directory. Any radix-8 integer (leading zeros may be omitted.)

Returns
true if the directory creation succeeds; false otherwise

Return type
bool

open(*path: str*, ***kwargs: str | int | None | List[str]*) → IO[bytes]

Return a file-like object for reading the given HDFS path.

Parameters

- **offset** (*long*) – The starting byte position.
- **length** (*long*) – The number of bytes to be processed.
- **buffersize** (*int*) – The size of the buffer used in transferring data.

Return type
file-like object

remove_xattr(*path: str*, *xattr_name: str*, ***kwargs: str | int | None | List[str]*) → None

Remove an xattr of a file or directory.

rename(*path: str*, *destination: str*, ***kwargs: str | int | None | List[str]*) → bool

Renames Path src to Path dst.

Returns
true if rename is successful

Return type
bool

rename_snapshot(*path: str*, *oldsnapshotname: str*, *snapshotname: str*, ***kwargs: str | int | None | List[str]*) → None

Rename a snapshot

set_owner(*path: str*, ***kwargs: str | int | None | List[str]*) → None

Set owner of a path (i.e. a file or a directory).

The parameters owner and group cannot both be null.

Parameters

- **owner** – user

- **group** – group

set_permission(*path: str*, ***kwargs: str | int | None | List[str]*) → None

Set permission of a path.

Parameters

permission (*octal*) – The permission of a file/directory. Any radix-8 integer (leading zeros may be omitted.)

set_replication(*path: str*, ***kwargs: str | int | None | List[str]*) → bool

Set replication for an existing file.

Parameters

replication (*short*) – new replication

Returns

true if successful; false if file does not exist or is a directory

Return type

bool

set_times(*path: str*, ***kwargs: str | int | None | List[str]*) → None

Set access time of a file.

Parameters

- **modificationtime** (*long*) – Set the modification time of this file. The number of milliseconds since Jan 1, 1970.
- **accesstime** (*long*) – Set the access time of this file. The number of milliseconds since Jan 1 1970.

set_xattr(*path: str*, *xattr_name: str*, *xattr_value: str | None*, *flag: str*, ***kwargs: str | int | None | List[str]*) → None

Set an xattr of a file or directory.

Parameters

- **xattr_name** – The name must be prefixed with the namespace followed by `.`. For example, `user.attr`.
- **flag** – `CREATE` or `REPLACE`

walk(*top: str*, *topdown: bool = True*, *onerror: Callable[[HdfsException], None] | None = None*, ***kwargs: str | int | None | List[str]*) → Iterator[Tuple[str, List[str], List[str]]]

See `os.walk` for documentation

exception pyhdfs.**HdfsDSQuotaExceededException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsQuotaExceededException*

exception pyhdfs.**HdfsException**

Bases: `Exception`

Base class for all errors while communicating with WebHDFS server

exception pyhdfs.**HdfsFileAlreadyExistsException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsIOException*

exception pyhdfs.**HdfsFileNotFoundException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsIOException*

exception pyhdfs.**HdfsHadoopIllegalArgumentException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsIllegalArgumentException*

exception pyhdfs.**HdfsHttpException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsException*

The client was able to talk to the server but got a HTTP error code.

Parameters

- **message** – Exception message
- **exception** – Name of the exception
- **javaClassName** – Java class name of the exception
- **status_code** (*int*) – HTTP status code
- **kwargs** – any extra attributes in case Hadoop adds more stuff

exception pyhdfs.**HdfsIOException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsHttpException*

exception pyhdfs.**HdfsIllegalArgumentException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsHttpException*

exception pyhdfs.**HdfsInvalidPathException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsHadoopIllegalArgumentException*

exception pyhdfs.**HdfsNSQuotaExceededException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsQuotaExceededException*

exception pyhdfs.**HdfsNoServerException**

Bases: *HdfsException*

The client was not able to reach any of the given servers

exception pyhdfs.**HdfsPathIsNotEmptyDirectoryException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsIOException*

exception pyhdfs.**HdfsQuotaExceededException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsIOException*

exception pyhdfs.**HdfsRemoteException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsIOException*

exception pyhdfs.**HdfsRetriableException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsIOException*

exception pyhdfs.**HdfsRuntimeException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsHttpException*

exception pyhdfs.**HdfsSecurityException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsHttpException*

exception pyhdfs.**HdfsSnapshotException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsIOException*

exception pyhdfs.**HdfsStandbyException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsIOException*

exception pyhdfs.**HdfsUnsupportedOperationException**(*message: str*, *exception: str*, *status_code: int*, ***kwargs: object*)

Bases: *HdfsHttpException*

class pyhdfs.**TypeQuota**(***kwargs: object*)

Bases: `_BoilerplateClass`

Parameters

- **consumed** (*int*) – The storage type space consumed.
- **quota** (*int*) – The storage type quota.

consumed: int

quota: int

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

O

P

Q

R

S

T

W