lazyfile

Release 0.1

Paul Moore

Sep 22, 2021

CONTENTS

1 Example **3**

1.1 Tutorial 4

1.2 LazyFile How To Guides 4

2 Indices and tables **7**

The `lazyfile` module is an implementation of an "on demand" file object in Python. It allows you to define how the data for a file is fetched, and it ensures that as the file is read, data is only requested when needed, and is cached so that the same block of data is never requested more than once.

This is useful when requesting data is costly, for example when accessing a large file over HTTP, where range requests can be used to limit the amount of network traffic needed.

CHAPTER

ONE

EXAMPLE

We will demonstrate the library by implementing a HTTP-based file, that only requests data from the server as needed.

```
from lazyfile import LazyFile
from urllib.request import Request, urlopen
from functools import partial

# We need to know the size of the file when we
# create the LazyFile object, so we issue a
# HTTP HEAD request to get it
def content_len(url):
    req = Request(url, method="HEAD")
    with urlopen(req) as f:
        return int(f.headers["Content-Length"])

# Get a block of bytes from the URL.
def getter(url, lo, hi):
    # The library passes the start and one-past-the-end values
    # for the range, just like a Python slice. So we need to
    # adjust the end value for the Range header.
    req = Request(url, headers={"Range": f"bytes={lo}-{hi-1}"})

    # Read the actual data
    with urlopen(req) as f:
        data = f.read()
        assert len(data) == hi-lo, f"Data ({lo}, {hi}) = {data!r}"

    return data

def open_url(url):
    url_getter = partial(getter, url)
    file_size = content_len(url)
    return LazyFile(file_size, url_getter)
```

1.1 Tutorial

1.1.1 Basic Usage

You need to know two things to create a `LazyFile`. The length of the file, and how to get a block of data from it.

```
>>> from lazyfile import LazyFile
>>> lf = LazyFile(size, reader)
```

The size is just an integer. The `reader` argument is a function which will be passed the range of bytes to read, expressed as the index of the start of the range, and the index one past the end of the range (this is the same convention as for Python ranges). The indices will always be between 0 and the size of the file.

As a simple example, the following will implement a reader for a byte string:

```
>>> data = b"Hello, world!"
>>> def get_bytes(lo, hi):
...     return data[lo:hi]
...
>>> lf = LazyFile(len(data), get_bytes)
>>> lf.seek(3)
3
>>> lf.read(4)
b'lo, '
```

Obviously, this is pointless, as `io.BytesIO` does the job far more effectively. The `LazyFile` class is intended for use when the `get_bytes` function is expensive, and it is worth minimising the number of calls to it. But the above is fully functional, and demonstrates the usage without needing a more complex data source.

1.2 LazyFile How To Guides

1.2.1 How to lazily extract a file from a remote wheel

The original motivation for this library was to extract the metadata file from a wheel, without downloading the whole wheel file (which could potentially be very large).

Wheel files are structured as zip files, and can be read using the standard library `zipfile` module. The `ZipFile` constructor takes a file-like object which must return bytes, and be seekable. A LazyFile is ideal for this.

First, we need to implement methods to get the length of the file, and to get a block of bytes from the file.

```
from urllib.request import Request, urlopen

# Get the file size with a HEAD request
def content_len(url):
    req = Request(url, method="HEAD")
    with urlopen(req) as f:
        return int(f.headers["Content-Length"])

# Get a block of bytes from the URL.
def get_url_range(url, lo, hi):
    # Adjust the range, as hi is "past the end"
```

(continues on next page)

(continued from previous page)

```
    req = Request(url, headers={"Range": f"bytes={lo}-{hi-1}"})
    with urlopen(req) as f:
        data = f.read()
        if len(data) != hi-lo:
            raise ValueError(f"Failed to read {hi-lo} bytes")

    return data
```

With these helpers, we can open a URL lazily

```
from lazyfile import LazyFile
from functools import partial

def open_url(url):
    url_getter = partial(get_url_range, url)
    file_size = content_len(url)
    return LazyFile(file_size, url_getter)
```

And that's all we need to process the file as a zipfile and extract the metadata

```
def extract_metadata(url)
    f = open_url(url)
    z = ZipFile(f)
    for name in z.namelist():
        if name.endswith(".dist-info/METADATA"):
            metadata = z.read(name)
            return metadata
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search