Python实现Tus Client库的总结

一、Tus协议的Client端库整体介绍

Python的Client库是一个官方版本库，链接：https://github.com/tus/tus-py-client 。

对于这个库的基本使用，在README文件中有所介绍，这里简略介绍一下：

这里是初始化TusClient的类，第一个就是上传的服务器的地址。 如果需要，配置 Authorization 的 headers，不过目前使用，这个字段是必须的。

```
my_client = client.TusClient('http://master.tus.io/files/',
                             headers={'Authorization': 'Basic
xxyyZZAAbbCC='})
```

配置其他 headers，格式如下

```
my_client.set_headers({'HEADER_NAME': 'HEADER_VALUE'})
```

其实TusClient里面已经定义了uploader()函数，返回一个Uploader类，所以我们才可以如此使用

```
uploader = my_client.uploader('path/to/file.ext', chunk_size=200)
```

文件流可以替代文件路径，代码如下：

```
fs = open('path/to/file.ext')
uploader = my_client.uploader(file_stream=fs, chunk_size=200)
```

至于上传，那么我们只要执行

```
uploader.upload()
```

即可。

代码完整样例如下：

```
from tusclient import client

my_client = client.TusClient('http://master.tus.io/files/', headers=
{'Authorization': 'Basic xxyyZZAAbbCC='})
my_client.set_headers({'HEADER_NAME': 'HEADER_VALUE'})
```

```
uploader = my_client.uploader('path/to/file.ext', chunk_size=200)
uploader.upload()
```

其实更简单的代码也可以，直接调用Uploader也可，但是必须保证headers不是必须，而且上传的url是已知的才可以：

```
from tusclient.uploader import Uploader

my_uploader = Uploader('path/to/file.ext',
                       url='http://master.tus.io/files/abcdef123456',
                       chunk_size=200)
```

二、Uploader类详解

我们这里重点分析下上传类 Uploader 的函数，真正处理上传的函数都在这里。

1、参数总结：

- file_path 所需要上传文件的路径
- file_stream 所需要上传文件流，这个和file_path选择填一个即可，推荐用file_path
- url 可选，这就是上传的url，但是如果在Client类配置了，这里可以不填
- client 可选，就是Client类的接收
- chunk_size 可选，chunk_size的大小
- metadata 可选，填充上传的metadata信息
- retries 可选，每个上传允许的重试次数
- retry_delay 可选，每两次重试的间隔
- store_url 可选，布尔型，是否储存url，其实就是是否允许断点
- url_storage 可选，url的储存结构，这个和store_url必须同时填充才可以断点，库里提供了Storage这个接口，实现Storage方法即可返回给url_storahe；同时库里提供了一个采用TinyDB的FileStorage的实例，可以参考
- fingerprinter 可选，库里原本提供了Fingerprint接口，实现生成独有的和文件匹配的key，用来实现key:url键值对储存用的；同时库里提供了Fingerprint类实现了这个接口，可以参考
- log_func 可选，输出log日志的函数

2、主要函数分析

在Uploader里执行初始化时，就已经执行了几个函数：

- get_url()
- get_offset() 这两个函数自然是一个获取url的值，一个获取已上传量的值。

下面这个就是上传函数，是我们真正需要调用的函数，执行这个函数就执行上传了。

```
Uploader.upload()
```

那么在`Uploader.upload()`函数内部，实际是在执行`Uploader.upload_chunk()`函数，在这个函数里，调用了`_do_request`这个内部函数，这个内部函数调用了另一个类 TusRequest 里的HTTP的Patch上传请求，来实现上传。对于TusRequest类，此处不再赘述。

我们这里来看两个内部的函数：

```
def _do_request(self):
    # TODO: Maybe the request should not be re-created everytime.
    # The request handle could be left open until upload is done instead.
    self.request = TusRequest(self)
    try:
        self.request.perform()
        self.verify_upload()
    except TusUploadFailed as error:
        self.request.close()
        self._retry_or_cry(error)
    finally:
        self.request.close()
```

这里采用了try...except...finally来执行。`self.request.perform()`和`self.verify_upload()`是分别执行上传文件和确认是否上传完成的函数。其中确认是否上传完成函数是根据返回的 status code是否为204来判断的，如果不是204，那么自然抛出TusUploadFailed错误，然后执行except里的函数，关闭连接，然后重试连接。如果except不再执行了，自然就是finally里的关闭请求了。

注意，这个库的作者给我们留了个TODO，其实是说目前这个库里实现的上传方法是出现问题就关闭请求再次尝试连接，我们可以实现让请求一直开放直到上传完成，这个地方如果有可能可以自己尝试看看来修改下源码实现该功能。

这就是`_retry_or_cry()`这个内部函数，名字挺搞笑的，也就是要么重试要么哭，失败了就哭……这里的判定依据其实就是retries这个重试次数是否达到上限，如果达到了，就直接抛出error不再上传了。

```
def _retry_or_cry(self, error):
    if self.retries > self._retried:
        time.sleep(self.retry_delay)

        self._retried += 1
        try:
            self.offset = self.get_offset()
        except TusCommunicationError as e:
            self._retry_or_cry(e)
        else:
            self._do_request()
    else:
        raise error
```

在执行上传之前，chunk_size可以如此修改，这是uploader类里的self变量 uploader.chunk_size = 800
uploader.upload()

Continue uploading chunks till total chunks uploaded reaches 1000 bytes. uploader.upload(stop_at=1000)

三、TusClient类

其实这个类里没有什么内容，初始化参数就是两个

- url
- headers 一个是服务器的地址，一个是所需要的headers。

```
    def set_headers(self, headers):
        self.headers.update(headers)

    def uploader(self, *args, **kwargs):
        kwargs['client'] = self
        return Uploader(*args, **kwargs)
```

里面只有两个函数，一个是为后面添加headers使用，另一个就是为我们初始化并且返回了一个Uploader的类。

四、TusRequest类

这个类没有多少需要我们更改的，我们能知道这个类其实就是处理HTTP请求的即可。