Streamable log in browser

Is it possible to have a stream of log records in your browser just like you get with `tail -f`? Yes it is and it's not hard to do so at all.

Usage

This technique can be used for example in a case where you have a long running action and you want to give user a feedback what the progress is and what's actually going on - not just a progress bar with percentages.

Simplest example with built-in Python server

Python provides HttpServer and BaseHTTPRequestHandler classes. The first one is a primitive HTTP server, that can accept an HTTP request and hand it over to the second class `BaseHTTPRequestHandler` which processes it and returns an HTTP response.

Base on this one can write a handler that can execute an expensive operation that logs through logging module. In order to "catch" the logs a new logger needs to be introduced. Once we can catch the logs we can stream into browser.

New logging handler

Very simple handler that extends `logging.StreamHandler` and hands over every log that comes into the given stream (`wfile` here).

```
class BrowserStreamHandler(logging.StreamHandler):
    def __init__(self, wfile):
        super().__init__()
        self.wfile = wfile

    def emit(self, record):
        msg = self.format(record)
        self.wfile.write(f'{msg}\n'.encode())
        self.wfile.flush()
```

Primitive HTTP request handler

A simple HTTP handler that accepts GET requests, sets headers, adds the `BrowserStreamHandler` handler to root logger, performs a long running operation that utilizes logging and removes the handler once it's done.

```
class Server(BaseHTTPRequestHandler):
    def do_GET(self):
        try:
            self.send_response(200)
            self.send_header('Content-type', 'text/plain')
            self.send_header('X-Content-Type-Options', 'nosniff')
            self.send_header('Cache-Control', 'no-cache')
            self.end_headers()

            # Setting up the logger.
            browser_handler = BrowserStreamHandler(self.wfile)
            browser_handler.setFormatter(
                logging.Formatter(
                    '%(asctime)s.%(msecs)06d - %(name)s - %(levelname)s -
                    ↪  %(message)s', '%Y-%m-%d %H:%M:%S'
                )
            )

            # Add logging handler to root logger to capture all messages.
            root_logger = logging.getLogger()
            root_logger.addHandler(browser_handler)

            try:
                response_data = long_running_op_with_logging(request)
            finally:
                root_logger.removeHandler(browser_handler)

        except Exception as e:
            logging.exception(e)

# Somewhere over the rainbow...
def long_running_op_with_logging(request):
    logging.info("log 1")
    time.sleep(1)
    logging.info("log 2")
    time.sleep(1)
    logging.info("log 3")
```

```
# Simple HTTP server to orchestrate the whole thing.
if __name__ == '__main__':
    server = HTTPServer(('0.0.0.0', 8000), Server)
    LOGGER.info(f'Server started .... listening on {ip}:{port}')
    server.serve_forever()
```

A few very important notes:

- header `Content-type` must be either `text/plain` or `text/html` otherwise the browser won't stream the incomming data but wait until it's all loaded
- in case of `text/html` (because you want to make your output nice) you need to provide valid HTML from the start
- `Cache-Control` header makes sure the browser won't ever serve previously cached stream

That's pretty much it. The handler is now able to send streams (thanks to logger that hands logging messages over to `wfile`, which is the handler output stream). Once you open the page in the browser, you can see lines of logs which appear after 1 second. Looks like a magic without a single line of Javascript.

Django implementation

To implement this mechanism in Django framework one has to realize one important thing - Django doesn't provide a stream that logging handler can stream directly to. Instead of that, one has to create a queue, where a new handler will send logs to and a listener (which is a generator for StreamingHttpResponse) will pull those logs out from the queue and feed the streaming response.

On top of that, execution of `long_running_op_with_logging()` method must happen somewhere else (meaning threads) because the whole code is synchronous/blocking so the code would wait to finish the job first and then process the logging -> no streaming.

Once the long running operation is done a clean up is needed.

1. send `None` to the queue to break infinite `while` loop
2. close (join) the thread
3. send the rest of HTML so the page is complete in browser
4. remove handler from logger

Here is complete Django `views.py` (except the `long_running_op_with_logging()`)

```
import logging
import queue
import threading
```

```
from django.http import StreamingHttpResponse
from django.views.generic import View

class QueueLogHandler(logging.Handler):
    """
    A logging handler utilizing a queue - every log message
    it put into the queue.
    """
    def __init__(self, log_queue):
        super().__init__()
        self.log_queue = log_queue

    def emit(self, record):
        self.log_queue.put(self.format(record))


class ImportClientView(View):
    def get(self, request, *args, **kwargs):
        # 1. setup streaming logging.
        log_queue = queue.Queue()
        queue_handler = QueueLogHandler(log_queue)
        formatter = logging.Formatter('%(asctime)s - %(levelname)s -
↪   %(message)s')
        queue_handler.setFormatter(formatter)
        logger = logging.getLogger()
        logger.addHandler(queue_handler)
        logger.setLevel(logging.INFO)

        # 2. run the command in a separate thread.
        def run_command_and_cleanup():
            try:
                long_running_op_with_logging()
            except Exception:
                logger.error('Error calling importclient command',
↪   exc_info=True)
            finally:
                # Send None to stop streaming.
                log_queue.put(None)
                logger.removeHandler(queue_handler)

        thread = threading.Thread(target=run_command_and_cleanup)
        thread.start()
```

```
        # 3. Start streaming the response to the client.
        response = StreamingHttpResponse(self.generate_response(log_queue,
↪   thread), content_type='text/html')
        response['Cache-Control'] = 'no-cache'

        return response

    def generate_response(self, log_queue, thread):
        yield '<!DOCTYPE html><html><head>'
        yield "<meta charset='utf-8'>"
        yield '<title>Streaming Logs</title>'
        yield '</head><body><pre>'
        yield 'Starting import...\n'

        while True:
            # Blocks until next log message appears.
            record = log_queue.get()

            # Stop streaming on None message.
            if record is None:
                break

            yield f'{record}\n'

        # Join thread.
        thread.join()

        yield 'Import finished.\n'
        yield '</pre></body></html>'
```