Streaming Contents

Sometimes you want to send an enormous amount of data to the client, much more than you want to keep in memory. When you are generating the data on the fly though, how do you send that back to the client without the roundtrip to the filesystem?

The answer is by using generators and direct responses.

Basic Usage

This is a basic view function that generates a lot of CSV data on the fly. The trick is to have an inner function that uses a generator to generate data and to then invoke that function and pass it to a response object:

```
from flask import Response

@app.route('/large.csv')
def generate_large_csv():
    def generate():
        for row in iter_all_rows():
            yield ','.join(row) + '\n'
    return Response(generate(), mimetype='text/csv')
```

Each `yield` expression is directly sent to the browser. Note though that some WSGI middlewares might break streaming, so be careful there in debug environments with profilers and other things you might have enabled.

Streaming from Templates

The Jinja2 template engine also supports rendering templates piece by piece. This functionality is not directly exposed by Flask because it is quite uncommon, but you can easily do it yourself:

```
from flask import Response

def stream_template(template_name, **context):
    app.update_template_context(context)
    t = app.jinja_env.get_template(template_name)
    rv = t.stream(context)
    rv.enable_buffering(5)
    return rv
```

```
@app.route('/my-large-page.html')
def render_large_template():
    rows = iter_all_rows()
    return Response(stream_template('the_template.html', rows=rows))
```

The trick here is to get the template object from the Jinja2 environment on the application and to call **`stream()`** instead of **`render()`** which returns a stream object instead of a string. Since we're bypassing the Flask template render functions and using the template object itself we have to make sure to update the render context ourselves by calling **`update_template_context()`**. The template is then evaluated as the stream is iterated over. Since each time you do a yield the server will flush the content to the client you might want to buffer up a few items in the template which you can do with `rv.enable_buffering(size)`. `5` is a sane default.

Streaming with Context

▶ *Changelog*

Note that when you stream data, the request context is already gone the moment the function executes. Flask 0.9 provides you with a helper that can keep the request context around during the execution of the generator:

```
from flask import stream_with_context, request, Response

@app.route('/stream')
def streamed_response():
    def generate():
        yield 'Hello '
        yield request.args['name']
        yield '!'
    return Response(stream_with_context(generate()))
```

Without the **`stream_with_context()`** function you would get a **`RuntimeError`** at that point.