LIVING THE STREAM DREAM

By Ryan Ramage / @ryan_ramage_

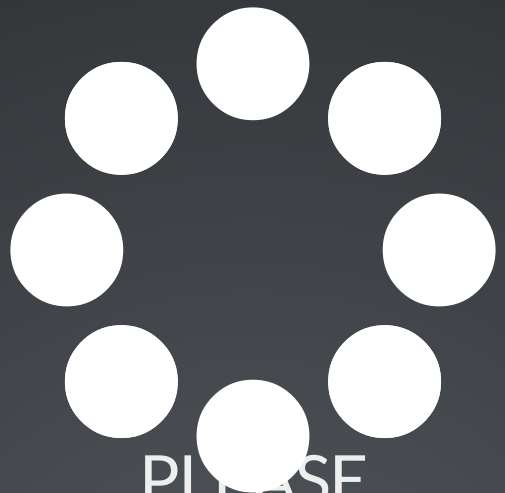
INTERACTIVE LIVE ACTION DEMO

JUST REMEMBER:

PEOPLE == DATA

TYPICAL SEARCH PATTERN

Search



PLEASE
WAIT

RESULT: ONE BIG TABLE

thing1	thing2	thing3
1	2	3
1	2	3
1	2	3
1	2	3
1	2	3
1	2	3
1	2	3
1	2	3
1	2	3

BETTER SEARCH PATTERN

Search

1 2 3
1 2 3
1 2 3
1 2 3
1 2 3
1 2 3
1 2 3
1 2 3

WHAT WE LEARNED

- Streaming decrease user idle time. [time]
- Streaming can lower memory requirements [space]

GTSTTUASAP PRINCIPLE

Get the shituff to the user as fast as possible

Because Speed is King in applications

STREAMS ARE GOOD ABSTRACTIONS

Garden hose

Dealing with disk/network/io/processor variability

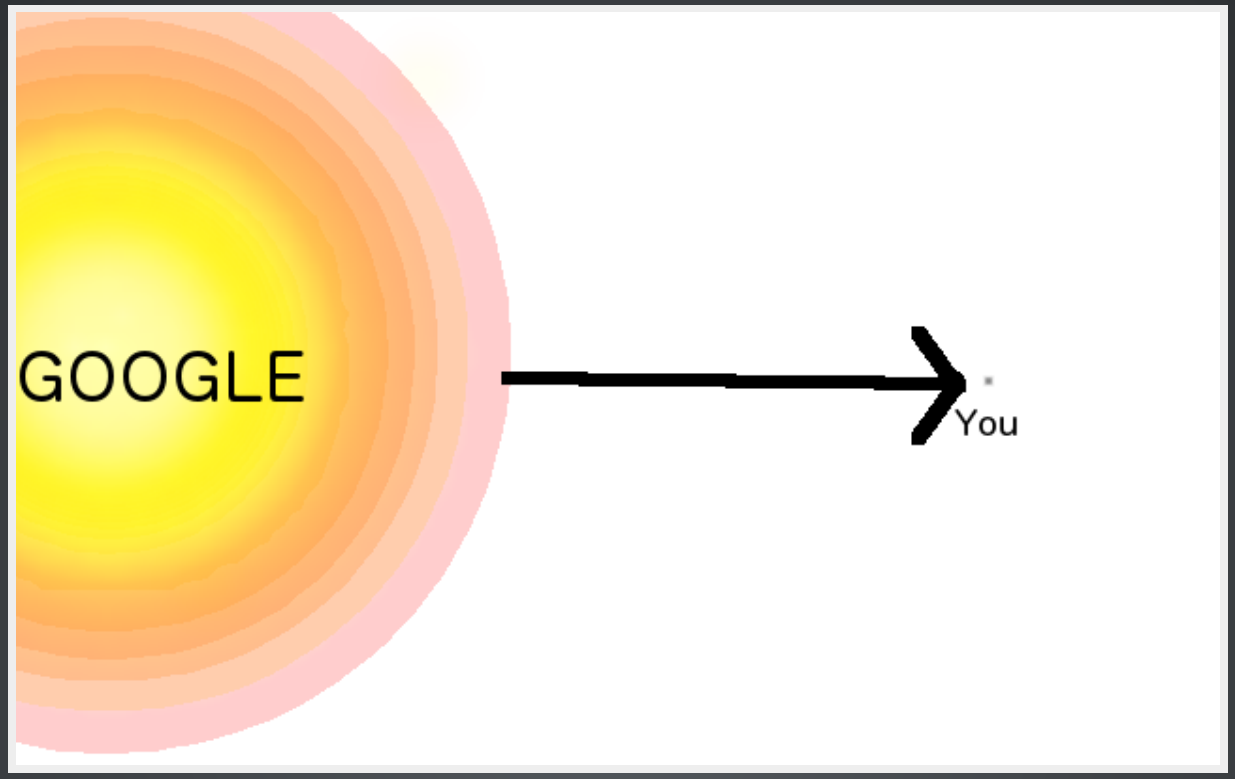
Keep memory spikes down

WHAT IS THE STREAM DREAM?

STREAMING SUPPORT ACROSS YOUR STACK

- Streaming Transport Level
- Streaming Browser Libs
- Streaming Middleware
- Streaming DB
- Streaming Build System (optional)

STREAMING TRANSPORT LEVEL


GOOGLE
You

TCP

The internet was built for streams.

> *TCP uses an end-to-end flow control protocol to avoid having the sender send data too fast for the TCP receiver to receive and process it reliably.*

-Wikipedia on TCP Flow control

THE STREAM DREAM?

STREAMING SUPPORT ACROSS YOUR STACK

- ~~Streaming Transport Level~~
- **Streaming Browser Libs**
- Streaming Middleware
- Streaming DB
- Streaming Build System (optional)

OBOE.JS

Wrapping xhr with a progressive streaming interface
A JSON parser that sits somewhere between SAX and DOM.

OBOE.JS

Dealing with failure mid request/response

OBOE.JS

Incremental loading, and browser caching

OBOE.JS

Easy JSON slice and dice

```
GET /myapp/things
```

```
{
   "foods": [
      {"name":"aubergine",    "colour":"purple"},
      {"name":"apple",        "colour":"red"},
      {"name":"nuts",         "colour":"brown"}
   ],
   "badThings": [
      {"name":"poison",       "colour":"pink"},
      {"name":"broken_glass", "colour":"green"}
   ]
}
```

```
MyApp.showSpinner('#foods');

oboe('/myapp/things')
   .node('foods.*', function( foodThing ){
      $.templateThing.append( foodThing.name + ' is ' + foodThing.colour );
   })
   .node('badThings.*', function( badThing ){
      console.log( 'Danger! stay away from ' + badThings.name );
   })
   .done( function(things){
                MyApp.hideSpinner('#foods');
      console.log( 'there are ' + things.foods.length + ' things you can eat
                   'and ' + things.nonFoods.length + ' that you shouldn\'t.'
   });
```

DUCK TYPING

```
oboe('/myapp/things')
   .node('{name colour}', function( foodObject ) {
      // I'll get called for every object found that
      // has both a name and a colour
   };
```

SHOE

Streaming between node and the browser

THE STREAM DREAM?

STREAMING SUPPORT ACROSS YOUR STACK

- ~~Streaming Transport Level~~
- ~~Streaming Browser Libs~~
- **Streaming Middleware**
- Streaming DB
- Streaming Build System (optional)

NODEJS

"Streams in node are one of the rare occasions when doing something the fast way is actually easier. SO USE THEM. not since bash has streaming been introduced into a high level language as nicely as it is in node."

-@dominictarr in his high level node style guide

```
var http = require('http');
var fs = require('fs');

var server = http.createServer(function (req, res) {
    fs.readFile(__dirname + '/movie.mp4', function (err, data) {
        res.end(data);
    });
});
server.listen(8000);
```

BAD

```
var http = require('http');
var fs = require('fs');

var server = http.createServer(function (req, res) {
    var stream = fs.createReadStream(__dirname + '/movie.mp4');
    stream.pipe(res);
});
server.listen(8000);
```

GOOD

COPY A FILE

```
var fs = require("fs");

// Read File
fs.createReadStream("input/people.json")
    // Write File
    .pipe(fs.createWriteStream("output/people.json"));
```

PIPE

All the different types of streams use `.pipe()` to pair inputs with outputs.

`.pipe()` is just a function that takes a readable source stream `src` and hooks the output to a destination writable stream `dst`:

```
src.pipe(dst)
```

PIPE

```
a.pipe(b);
b.pipe(c);
c.pipe(d);
```

`.pipe(dst)` returns `dst` so that you can chain together multiple `.pipe()` calls together, which is the same as:

```
a.pipe(b).pipe(c).pipe(d)
```

This is very much like what you might do on the command-line to pipe programs together:

```
a | b | c | d
```

UN-GZIPPING A FILE

```
var fs = require("fs");
var zlib = require("zlib");

// Read File
fs.createReadStream("input/people.csv.gz")
    // Un-Gzip
    .pipe(zlib.createGunzip())
    // Write File
    .pipe(fs.createWriteStream("output/people.csv"));
```

NODE MODULES FOR STREAMING

Advice: Watch for > 1 year inactivity.
Streams have changed a lot

Look for trusted names in the stream eco-system

- dominictarr
- substack
- maxogden
- juliangruber
- raynos

NODE MODULES FOR STREAMING

mikeal/request

You can stream any response to a file stream.

```
request('http://google.com/doodle.png').pipe(fs.createWriteStream('doodle.png
```

NODE MODULES FOR STREAMING

dominictarr/event-stream

The EventStream functions resemble the array functions, because Streams are like Arrays, but laid out in time, rather than in memory.

Example of event stream

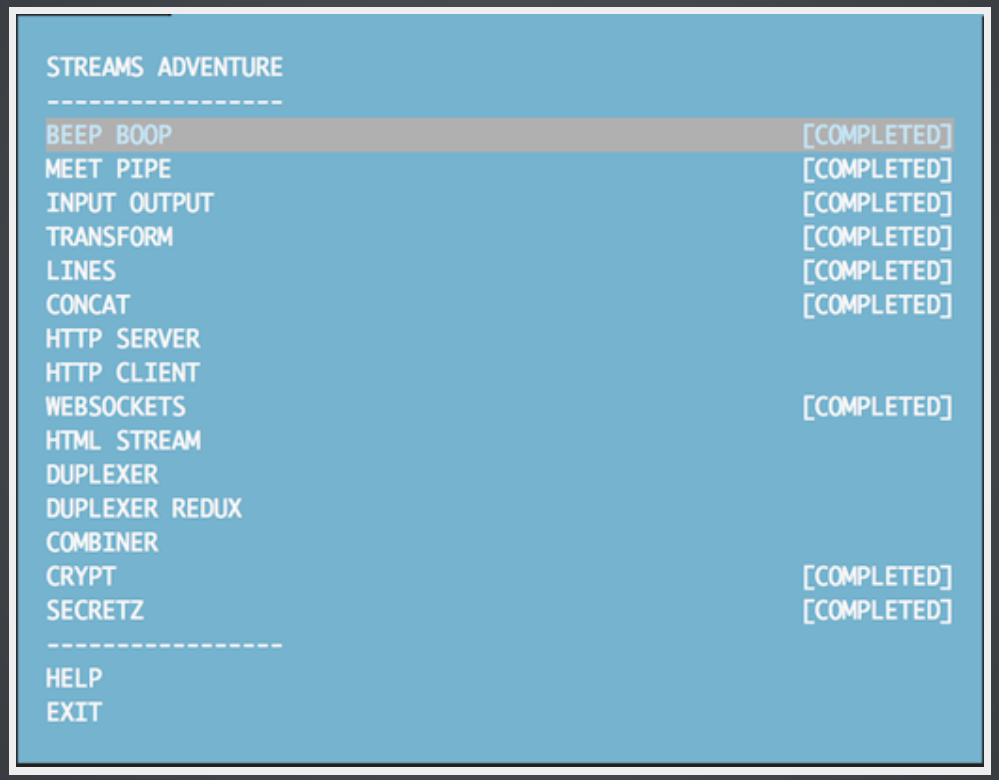
```
var inspect = require('util').inspect
var es = require('event-stream')    //load event-stream

es.pipe(                            //pipe joins streams together
  process.openStdin(),              //open stdin
  es.split(),                       //split stream to break on newlines
  es.map(function (data, callback) {//turn this async function into a stream
    var j
    try {
      j = JSON.parse(data)          //try to parse input into json
    } catch (err) {
      return callback(null, data)   //if it fails just pass it anyway
    }
    callback(null, inspect(j))      //render it nicely
  }),
  process.stdout                    // pipe it to stdout !
  )


// curl -sS registry.npmjs.org/event-stream | node pretty.js
//
```

STREAM ADVENTURE

```
> npm install -g stream-adventure
> stream-adventure
```



THE STREAM DREAM?

STREAMING SUPPORT ACROSS YOUR STACK

- ~~Streaming Transport Level~~
- ~~Streaming Browser Libs~~
- ~~Streaming Middleware~~
- **Streaming DB**
- Streaming Build System (optional)

COUCHDB

OF COURSE

STREAMABLE HTTP

```
request('http://localhost:5984/db/_design/app/_view/things_by_date').pipe(res
```

FILTER WITH NODE!

```
var filter = function(data, emit) {
      data.forEach(function(db){
          if (db.indexOf('dumb_user-') === 0) {
              emit('"' +  strip_prefix(db) + '"');
          }
      });
  }
  var filter_through = new FilterThrough(filter);
  request('http://localhost:5984/_all_dbs')
          .pipe(filter_through)
          .pipe(resp);
```

CONTINUOUS CHANGES

```
00:00: > curl -X GET "http://localhost:5984/db/_changes?feed=continuous&since
{"seq":4,"id":"test4","changes":[{"rev":"1-02c6b758b08360abefc383d74ed5973d"}
{"seq":5,"id":"test5","changes":[{"rev":"1-02c6b758b08360abefc383d74ed5973d"}
```

RVAGG/NODE-LEVELUP

RVAGG/NODE-LEVELUP

EVERYTHING STREAMS

```
var levelup = require('levelup');
var srcdb = levelup('./srcdb');
var dstdb = levelup('./destdb');

srcdb.put('name', 'LevelUP');

srcdb.createReadStream().pipe(dstdb.createWriteStream()).on('close', onDone)
```

RVAGG/NODE-LEVELUP

START/END KEYS

```
srcdb.createReadStream({
        start: 'n',
        end: 'k'
}).pipe(resp)
```

THE STREAM DREAM?

STREAMING SUPPORT ACROSS YOUR STACK

- ~~Streaming Transport Level~~
- ~~Streaming Browser Libs~~
- ~~Streaming Middleware~~
- ~~Streaming DB~~
- **Streaming Build System (optional)**

GULP - STREAMING BUILDS

SAMPLE GULPFILE

```
var scriptFiles = './src/**/*.js';

gulp.task('compile', function(){
  // concat all scripts, minify, and output
  gulp.src(scriptFiles)
    .pipe(concat({fileName: pkg.name+".js"})
    .pipe(minify())
    .pipe(gulp.dest('./dist/'));
});

gulp.task('test', function(){
  // lint our scripts
  gulp.src(scriptFiles).pipe(jshint());

  // run our tests
  spawn('npm', ['test'], {stdio: 'inherit'});
});

gulp.task('default', function(){
  gulp.run('test', 'compile');
  gulp.watch(scriptFiles, function(){
    gulp.run('test', 'compile');
  });
});
```

CONS

BECAUSE NOTHING IS FREE

MIGHT NOT PLAY WELL WITH YOUR 'FRAMEWORK'

MVC-RAP ROUTES AND FUNCTIONS

```
// please provide
class People extends MVCrap {
        getAll()
        getByX()
}
```

MUCH CONFUSE

stream-to-pull-stream: Convert a classic-stream, or a new-stream into a pull-stream

invert-stream: Create a pair of streams (A, B) such that A.write(X) -> B.emit('data', X) and B.write(X) -> A.emit('data', X)

BACKPRESSURE AND UNBOUNDED CONCURRENCY

The Node concurrency model is kind of like a credit card for computing work. Credit cards free you from the hassle and risks of carrying cash, (...) unless you spend more than you make. Try making 200,000 HTTP client requests in a tight loop, starting each request without waiting for the previous response.

BACKPRESSURE AND UNBOUNDED CONCURRENCY

Once a process can't keep up with its work and starts to get slow, this slowness often cascades into other processes in the system. Streams are not the solution here. In fact, streams are kind of what gets us into this mess in the first place by providing a convenient fire and forget API for an individual socket or HTTP request without regard to the health of the rest of the system

- Matt Ranney

BUT THE DREAM LIVES ON...

Mix and match:

- Oboe.js
- Node Streams
- CouchDB
- node-LevelDB
- gulp

Try them out today!


THANKS!
LIVING THE STREAM DREAM
By Ryan Ramage / @ryan_ramage_