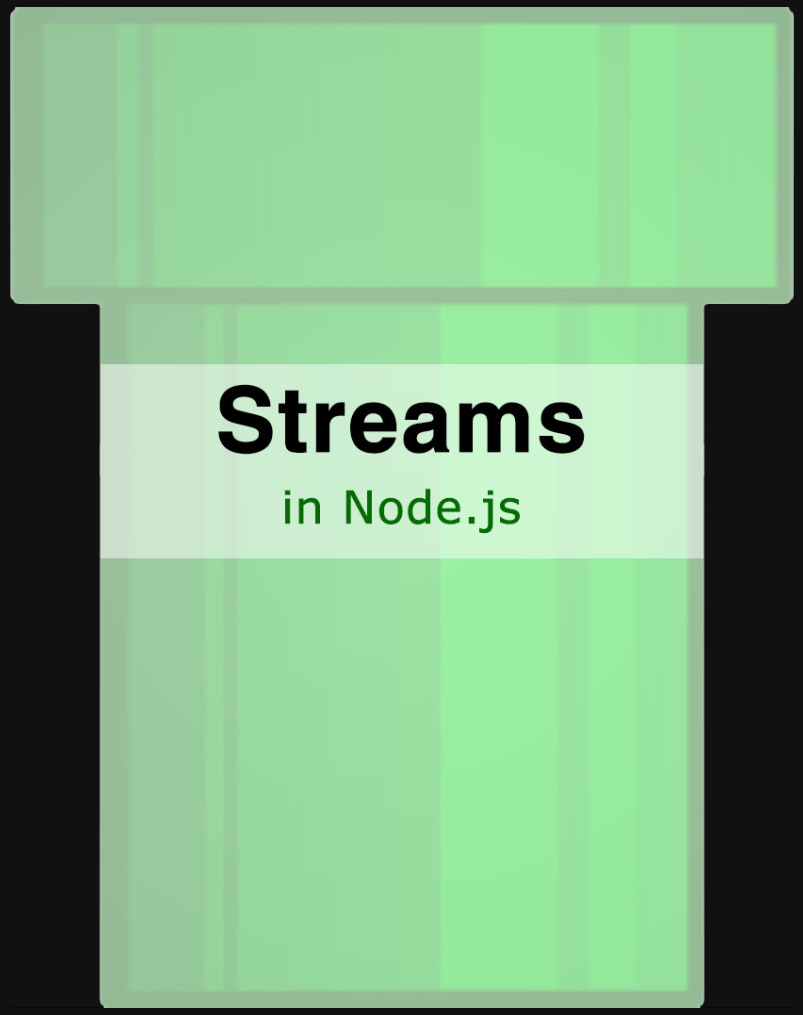
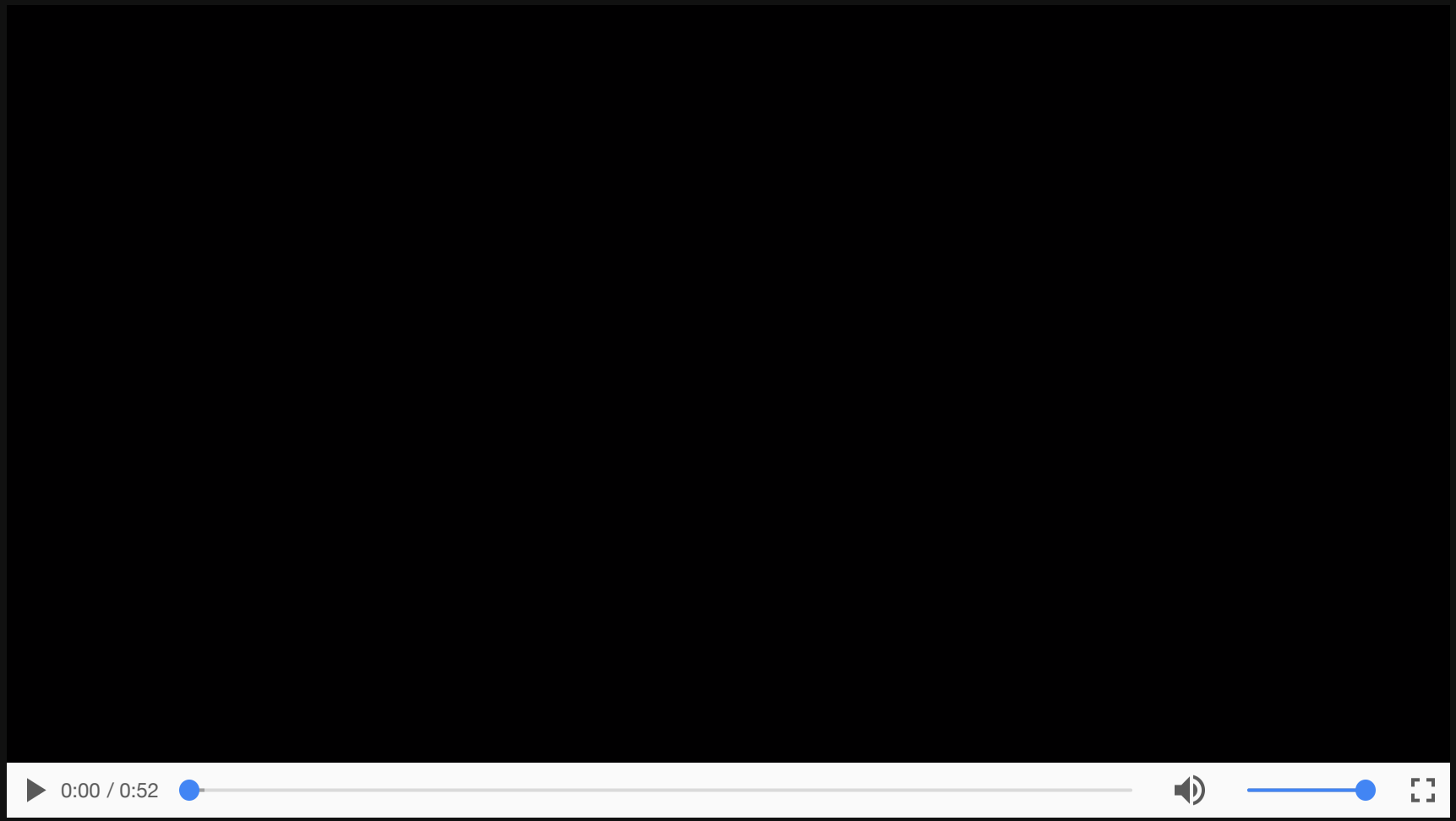
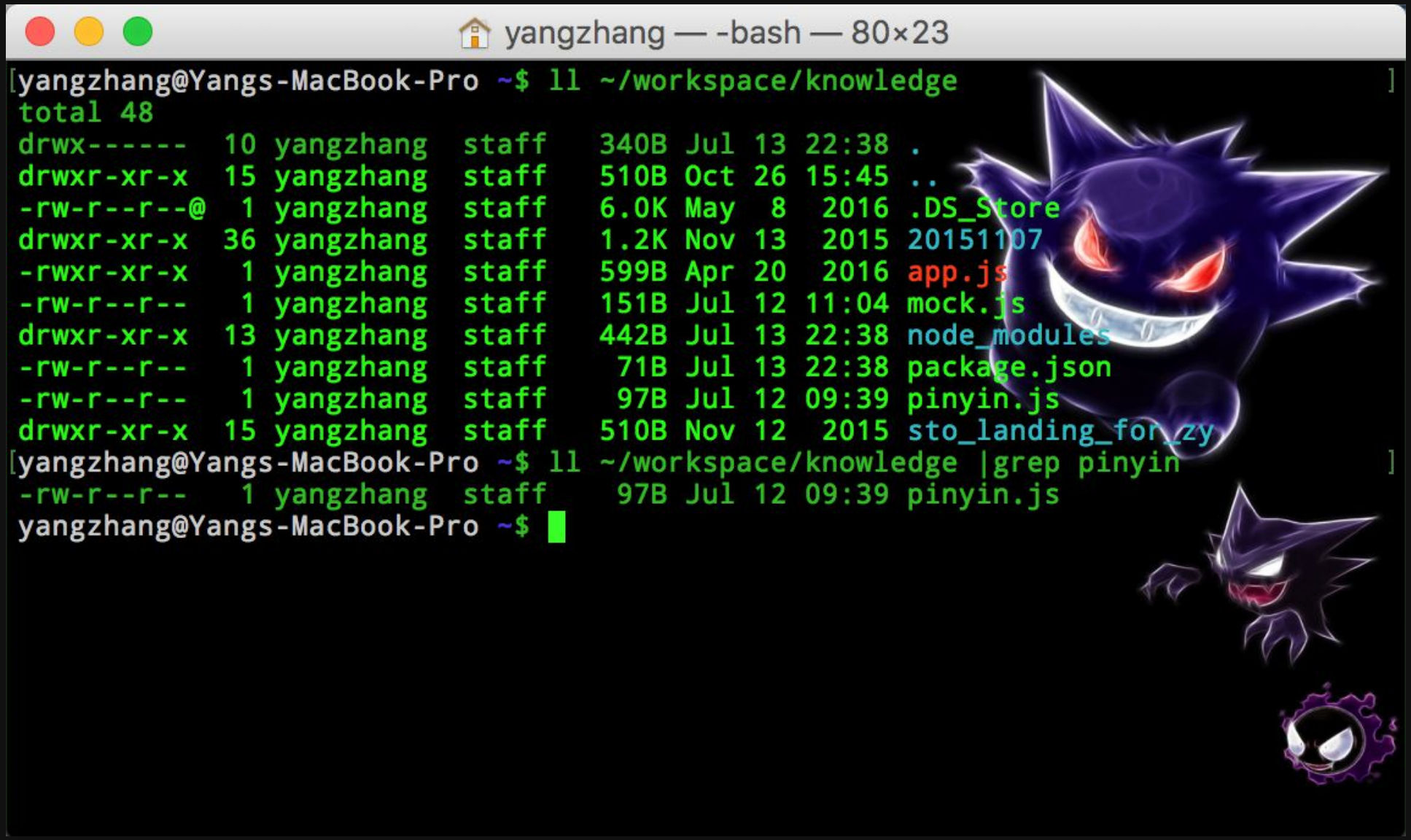


Created by Yang


0:07 / 0:52


0:00 / 0:52


yangzhang — -bash — 80×23
[yangzhang@Yangs-MacBook-Pro ~$ ll ~/workspace/knowledge]
total 48
drwx------ 10 yangzhang staff   340B Jul 13 22:38 .
drwxr-xr-x 15 yangzhang staff   510B Oct 26 15:45 ..
-rw-r--r--@ 1 yangzhang staff   6.0K May 8 2016 .DS_Store
drwxr-xr-x 36 yangzhang staff   1.2K Nov 13 2015 20151107
-rwxr-xr-x   1 yangzhang staff   599B Apr 20 2016 app.js
-rw-r--r--   1 yangzhang staff   151B Jul 12 11:04 mock.js
drwxr-xr-x 13 yangzhang staff   442B Jul 13 22:38 node_modules
-rw-r--r--   1 yangzhang staff   71B Jul 13 22:38 package.json
-rw-r--r--   1 yangzhang staff   97B Jul 12 09:39 pinyin.js
drwxr-xr-x 15 yangzhang staff   510B Nov 12 2015 sto_landing_for_zy
[yangzhang@Yangs-MacBook-Pro ~$ ll ~/workspace/knowledge |grep pinyin]
-rw-r--r--   1 yangzhang staff   97B Jul 12 09:39 pinyin.js
yangzhang@Yangs-MacBook-Pro ~$

什么是流

"We should have some ways of connecting programs like garden hose--screw in another segment when it becomes necessary to massage data in another way. This is the way of IO also."

Doug McIlroy. October 11, 1964

separation of concerns

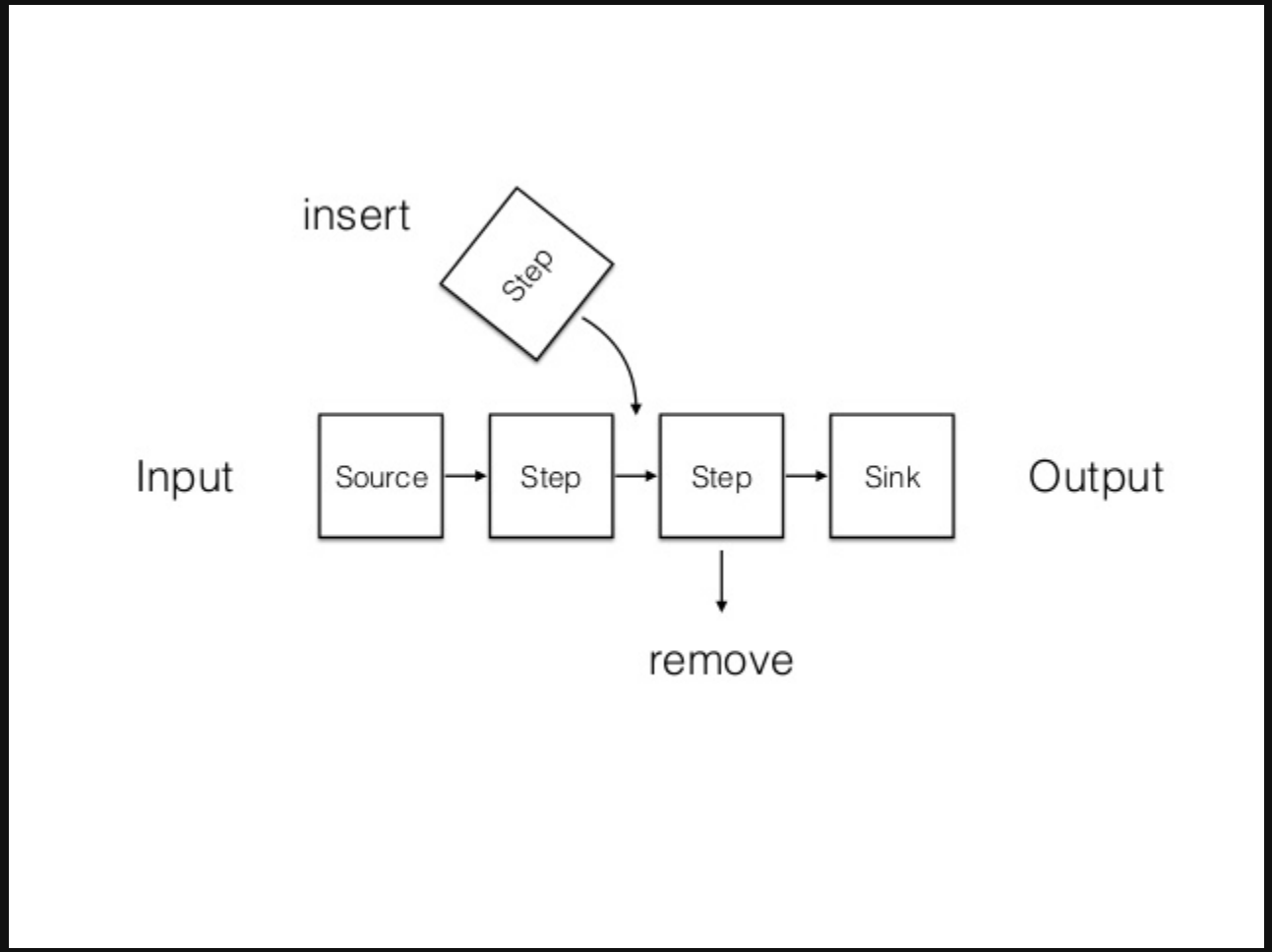
关注点分离

- Source

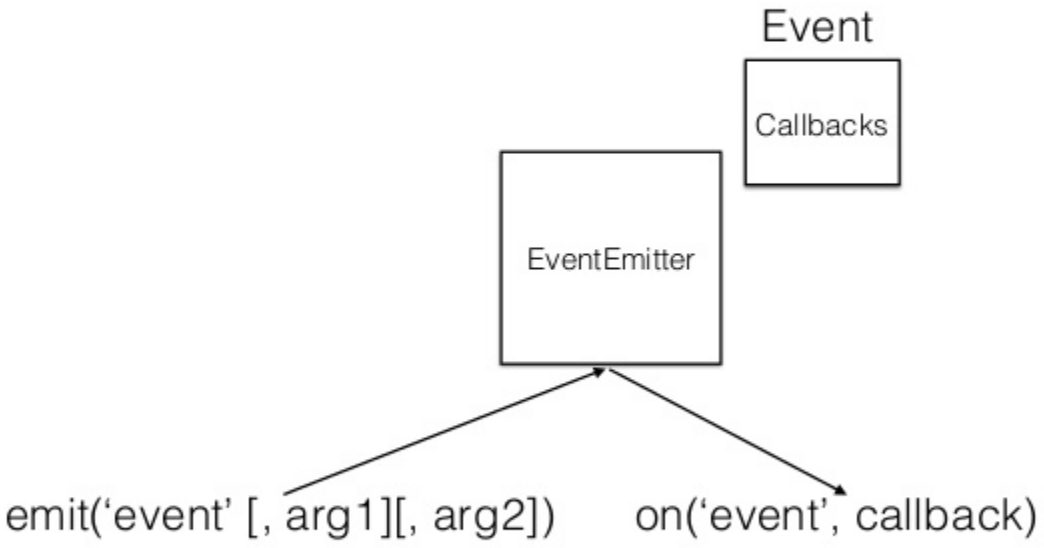
  - Where the data comes from.
- Pipeline

  - Where you filter or transform your data as it passes through.
- Sink

  - Where your data ultimately goes.


insert
Step
Input
Source
Step
Step
Sink
Output
remove

Streams are EventEmitters



为什么要使用流

- 更有效的使用内存
- 更有效的使用带宽
- 能够处理大量数据
- 关注点分离 SoC

适合使用流的场景

- http
- fs
- child_process
- tcp
- zlib
- crypto

- 视频特效
- 文件压缩
- 图片编码

Stream, Stream2 & Stream3

```
var Readable = require('stream').Readable || require('readable-stream').Readable
```

Push Streams

```
// node 0.8
var fs = require('fs');
var stream = fs.createReadStream('readme.txt');
stream.setEncoding('utf8');
stream.pause();

// Resume the stream in 1 second
setTimeout(stream.resume.bind(stream), 1000);

var data = '';
stream.on('data', function(chunk) {
  data += chunk;
})

stream.on('end', function() {
  // End of the stream has been reached and no more data can be read
  console.log('Data length: %d', data.length);
});
```

Pull Streams

```
// node 0.10 - pull stream, readable event example

// node 0.10 - push stream, data event example

// node 0.10 - Non-working mixing example
var fs = require('fs');
var stream = fs.createReadStream('readme.txt');
stream.setEncoding('utf8');
stream.pause();

var pulledData = '';
var pushedData = '';

stream.on('readable', function() {
  var chunk;
  while(chunk = stream.read()) {
    pulledData += chunk;
  }
});

stream.on('data', function(chunk) {
```

Combined Streams

```
// node 0.11+, io.js 1.0.1
var fs = require('fs');
var stream = fs.createReadStream('readme.txt');
stream.setEncoding('utf8');
stream.pause();

var pulledData = '';
var pushedData = '';

stream.on('readable', function() {
  var chunk;
  while(chunk = stream.read()) {
    pulledData += chunk;
  }
});

stream.on('data', function(chunk) {
  pushedData += chunk;
});

stream.on('end', function() {
```

through2

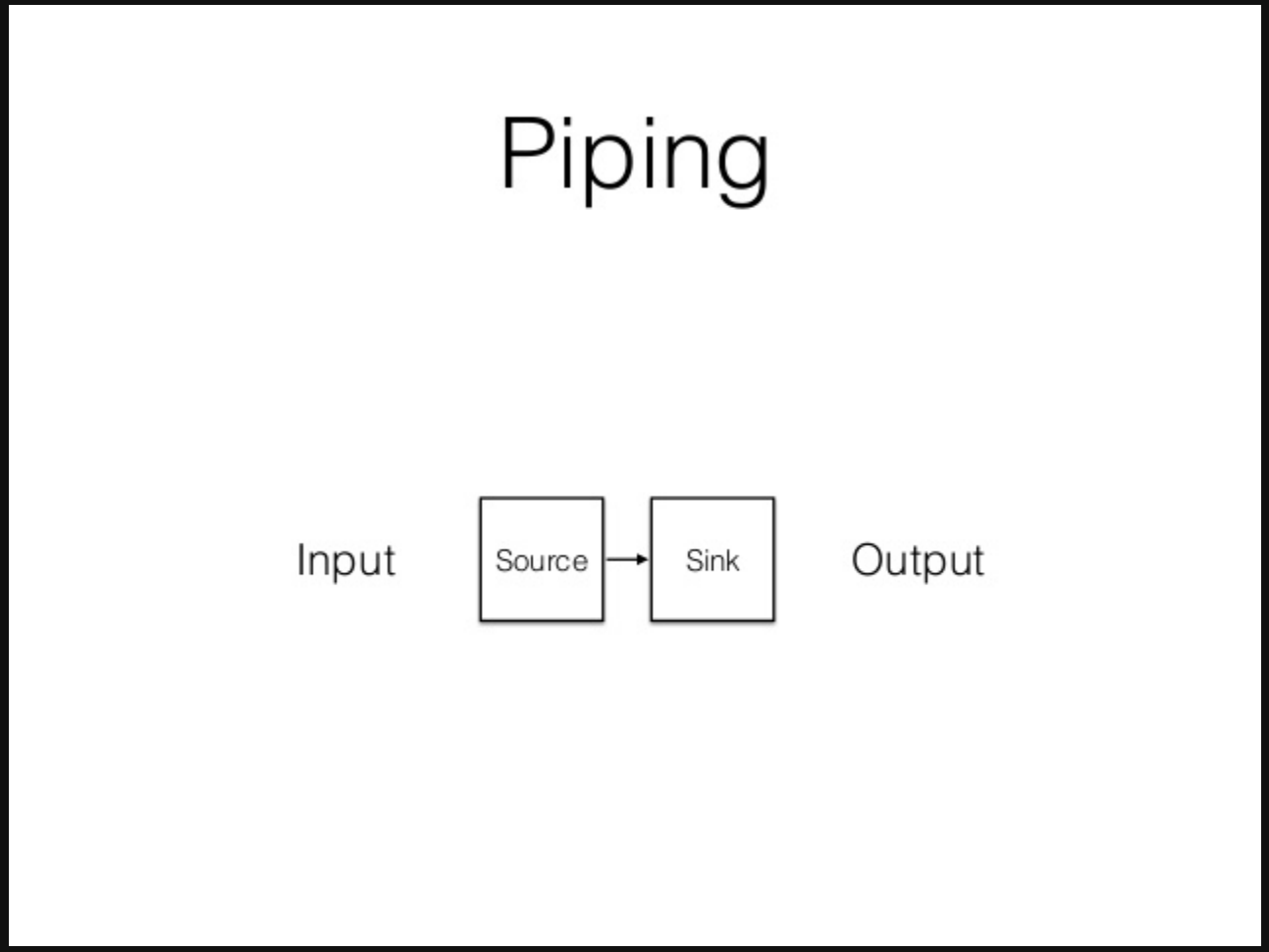
```
var through2 = require('through2')

fs.createReadStream('ex.txt')
  .pipe(through2(function (chunk, enc, callback) {

    for (var i = 0; i < chunk.length; i++)
      if (chunk[i] == 97)
        chunk[i] = 122 // swap 'a' for 'z'

    this.push(chunk)

    callback()

   }))
  .pipe(fs.createWriteStream('out.txt'))
```

四种类型的流

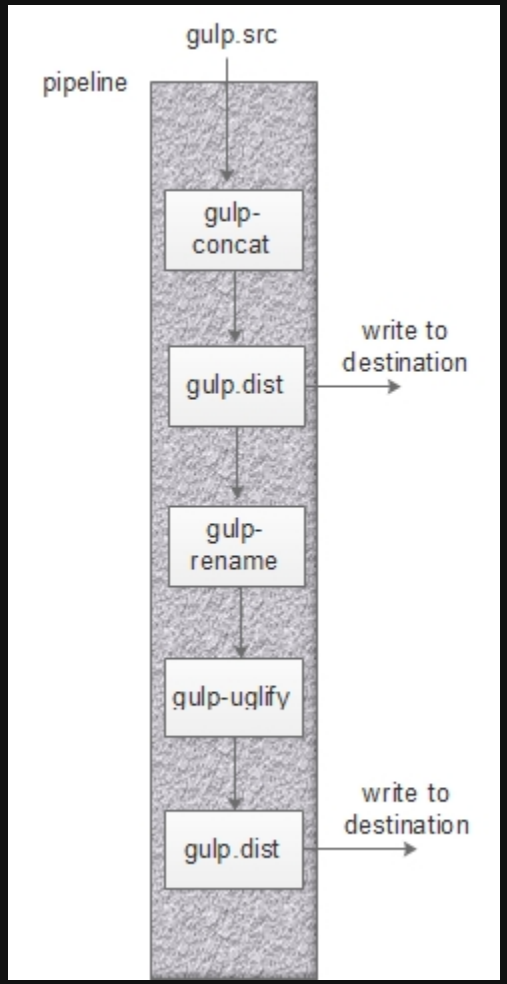
- Readable - sources
- Writable - sinks
- Duplex - both source and sink
- Transform - in-flight stream operations

```
ReadStream {
  connecting: false,
  _hadError: false,
  _handle:
   TTY {
     bytesRead: 20,
     _externalStream: {},
     fd: 13,
     writeQueueSize: 0,
     owner: [Circular],
     onread: [Function: onread],
     reading: true },
  _parent: null,
  _host: null,
  _readableState:
   ReadableState {
     objectMode: false,
     highWaterMark: 0,
     buffer: BufferList { head: null, tail: null, length: 0 },
     length: 0,
     pipes: null,
     pipesCount: 0,
     flowing: true,
     ended: false,
     endEmitted: false,
     reading: false,
     sync: false,
     needReadable: true,
     emittedReadable: false,
     readableListening: false,
     resumeScheduled: false,
     defaultEncoding: 'utf8',
     ranOut: false,
     awaitDrain: 0,
     readingMore: false,
     decoder: null,
     encoding: null },
  readable: true,
  domain: null,
  _events:
   { end: [ [Object], [Function: ontermend] ],
     finish: [Function: onSocketFinish],
     _socketEnd: [Function: onSocketEnd],
     pause: [Function],
     data: [Function: onData],
     keypress: [Function: onkeypress] },
  _eventsCount: 6,
```

管道



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

var read = fs.createReadStream('input.txt');
var write = fs.createWriteStream('output.txt');

write.on('pipe', function() {
    console.log('piping :D');
});

read.pipe(write);
```


gulp.src
pipeline
gulp-concat
write to destination
gulp.dist
gulp-rename
gulp-uglify
write to destination
gulp.dist

常见的各种流

build-in streams

process, child_process.spawn(), fs, net, http, zlib, crypto

control streams

through, from, pause-stream, concat-stream, duplex, duplexer, emit-stream, invert-stream, map-stream, remote-events, buffer-stream, event-stream, auth-stream

meta streams

mux-demux, stream-router, multi-channel-mdm

state streams

crdt, delta-stream, scuttlebutt, append-only

http streams

request, oppressor, response-stream

io streams

reconnect, kv, discovery-network

parser streams

tar, trumpet, jsonstream, json-scrape, stream-serializer

browser streams

shoe, domnode, sorta, graph-stream, arrow-keys, attribute, data-bind

html streams

hyperstream

audio streams

baudio

rpc streams

dnode, rpc-stream

test streams

tap, stream-spec

Gulp 里的流

- 任务调度：orchestrator
- 文件处理：vinyl-fs

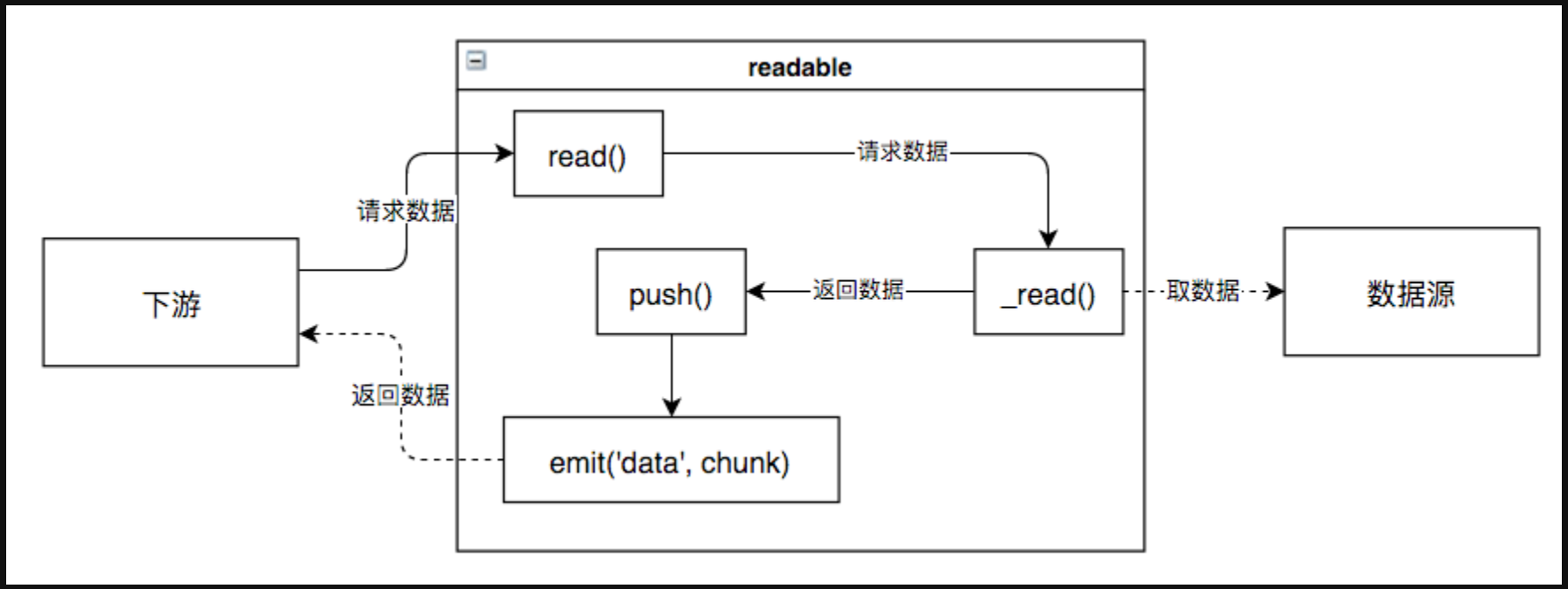
```
gulp.task('scripts', ['clean'], function() {
  // Minify and copy all JavaScript (except vendor scripts)
  // with sourcemaps all the way down
  return gulp.src(paths.scripts)
    .pipe(sourcemaps.init())
    .pipe(coffee())
    .pipe(uglify())
    .pipe(concat('all.min.js'))
    .pipe(sourcemaps.write())
    .pipe(gulp.dest('build/js'));
});
```

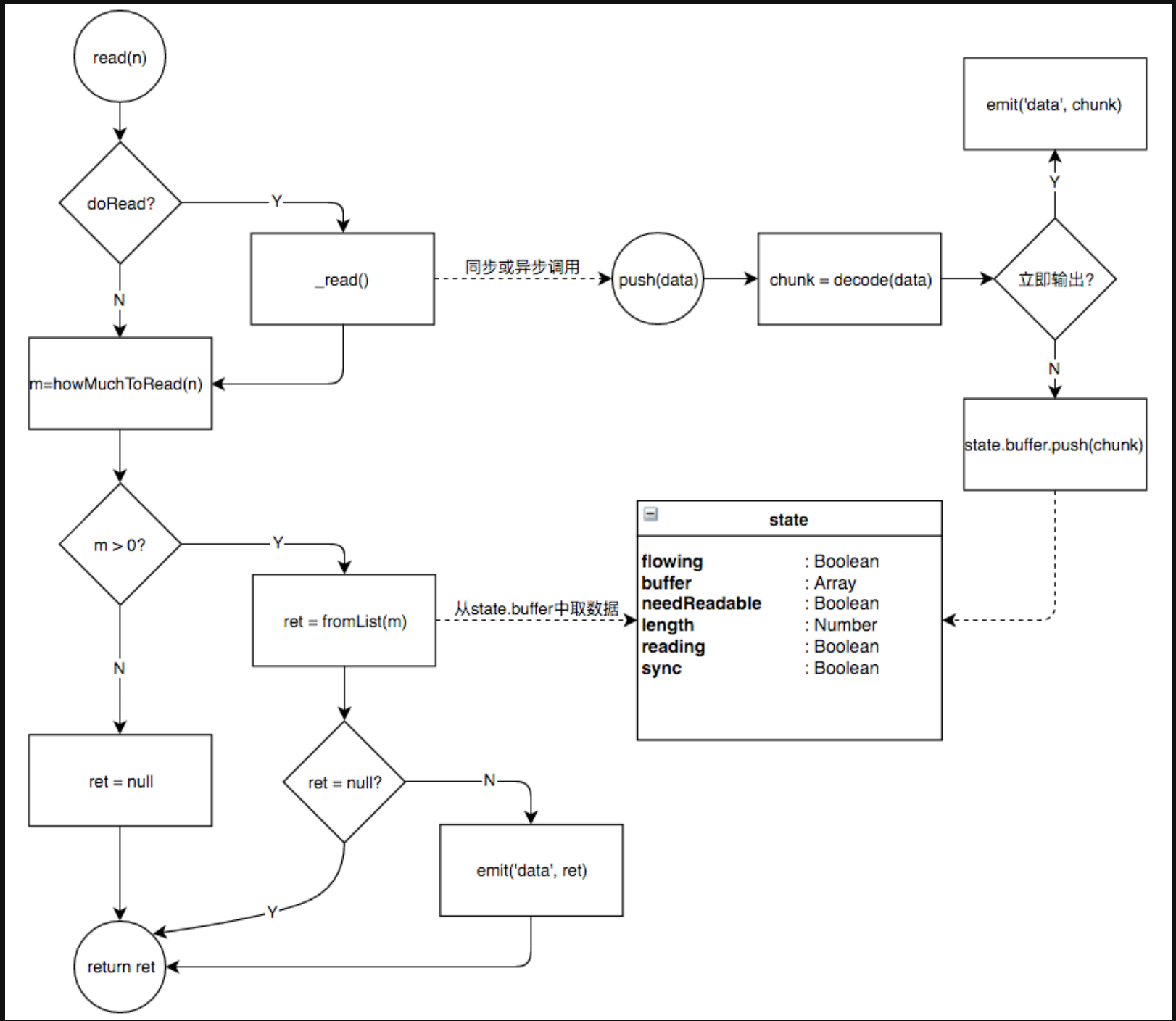
```
var Vinyl = require('vinyl');

var jsFile = new Vinyl({
  cwd: '/',
  base: '/test/',
  path: '/test/file.js',
  contents: new Buffer('var x = 123')
});
```

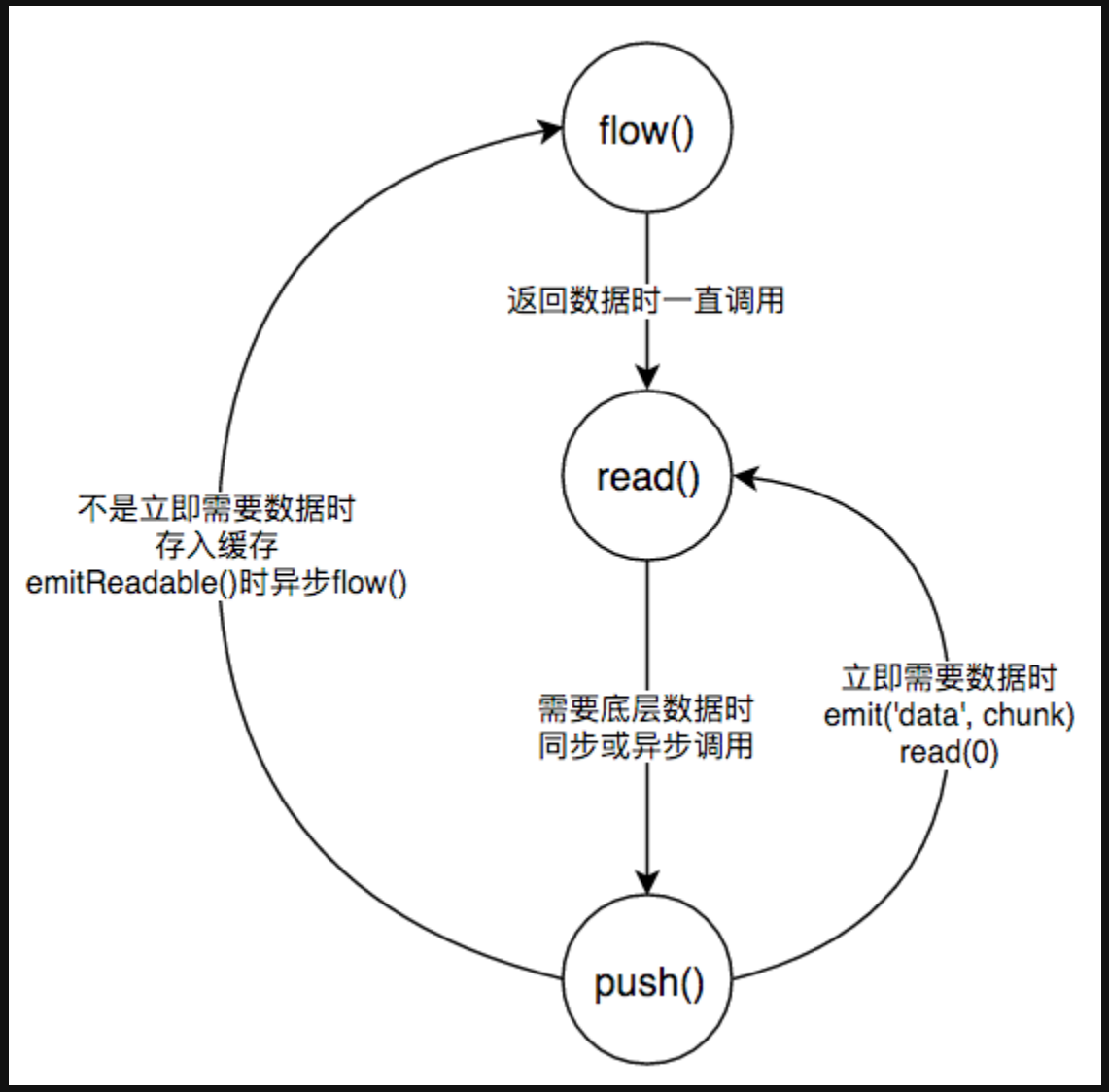
Vinyl is a very simple metadata object that describes a file.

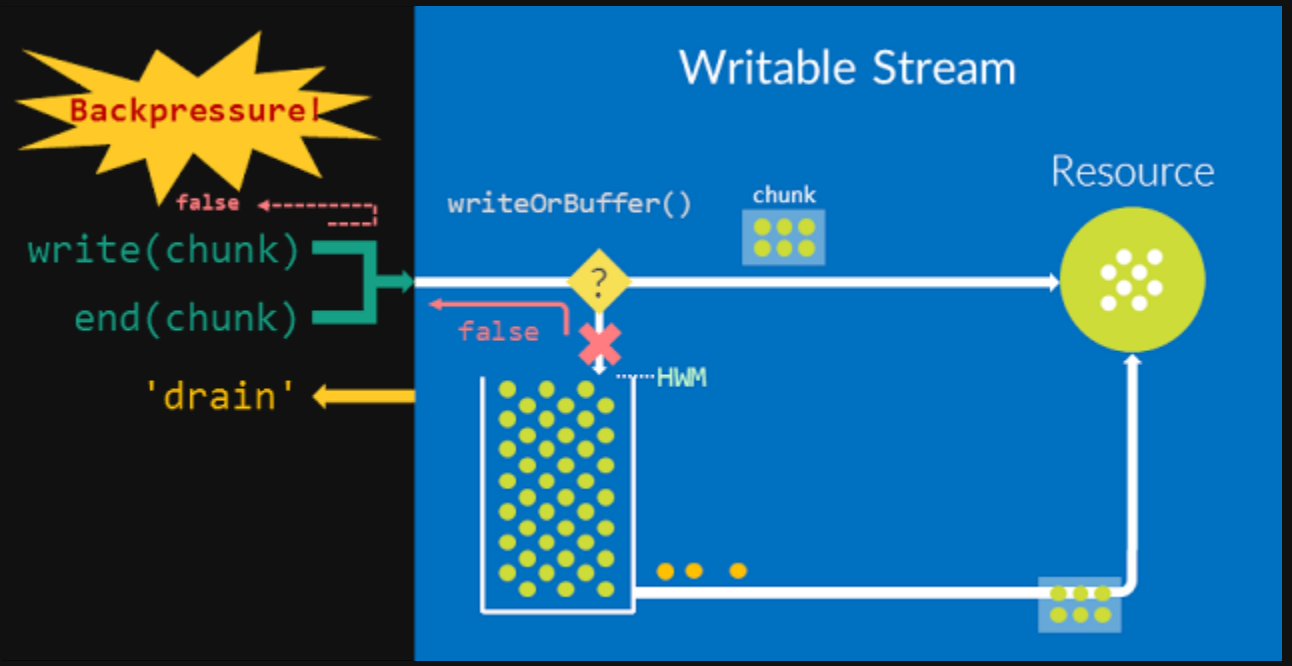
Vinyl-fs is a vinyl adapter for the file system.

流是如何工作的


readable
read()
请求数据
请求数据
下游
push()
返回数据
_read()
取数据
数据源
返回数据
emit('data', chunk)


read(n)
doRead?
Y
N
_read()
同步或异步调用
push(data)
chunk = decode(data)
立即输出?
Y
N
emit('data', chunk)
state.buffer.push(chunk)
m=howMuchToRead(n)
m > 0?
Y
N
ret = fromList(m)
从state.buffer中取数据
ret = null
ret = null?
N
Y
emit('data', ret)
return ret
state
flowing : Boolean
buffer : Array
needReadable : Boolean
length : Number
reading : Boolean
sync : Boolean


flow()
返回数据时一直调用
read()
不是立即需要数据时
存入缓存
emitReadable()时异步flow()
需要底层数据时
同步或异步调用
立即需要数据时
emit('data', chunk)
read(0)
push()


Backpressure!
false
write(chunk)
end(chunk)
'drain'
Writable Stream
writeOrBuffer()
chunk
Resource
?
false
HWM

如何实现自定义流

Use-case	Class	Method(s) to implement
Reading only	Readable	_read!
Writing only	Writable	_write!, _writev?
Reading and writing	Duplex	_read!, _write!, _writev?
Operate on written data, then read the result	Transform	_transform!, _flush?

```
var duplex = new stream.Duplex({
  read: function(n) {
    // sets this._read under the hood

    // push data onto the read queue, passing null
    // will signal the end of the stream (EOF)
    this.push(chunk);
  },
  write: function(chunk, encoding, next) {
    // sets this._write under the hood

    // An optional error can be passed as the first argument
    next()
  }
});
```

其实我想说的是

前端工程师的知识积累过程和方法

Reference

- Node.js Documentation
- Stream Handbook
- Node.js Stream - 基础、进阶、实战
- Cheatsheet and exemplary code
- Understanding Streams
- Why i don't use nodes core stream module
- streams spec from WHATWG

Thanks :D