Data Transform in Node.js Stream

- Yong Fei

Types of Stream

- Readable Stream
  - Ie: fs.createReadStream(filepath)
- Writable Stream
  - Ie: fs.createWriteStream(filepath)
- Duplex Stream
  - Ie: new net.Socket()
- Transform Stream
  - Ie: zlib.createGunzip()

Example: Gunzip file

```
// Create a Readable stream
const read = fs.createReadStream(sourcePath)

// zlib.createGunzip() returns a Transform stream that gunzip the data it receives
const transform = zlib.createGunzip()

// Create a Writable Stream
const write = fs.createWriteStream(targetPath)
```

Example: Gunzip file

```
// Stream is a type of EventEmitter.
// Attach a event listeners
read.on('end', () => console.log('No more data'))

transform.on('data', (data) => console.log(data.toString('utf-8')))

write.on('finish', () => console.log('Done writing to file.'))

// Start pushing the data
read.pipe(transform).pipe(write)
```

Example: Line break in stream

```
const split = require('split2')

// Create a readable stream
const reader = fs.createReadStream(sourcePath)

// Pipe data to a transform stream created with `split2` package.
const stream = reader
 .pipe(zlib.createGunzip())
 .pipe(split())
```

Example: Line break in stream

```
const stream = reader
 .pipe(zlib.createGunzip())
 .pipe(split())

// The .pipe() method returns a reference to the destination stream
// making it possible to set up chains of piped streams.
// So `stream` is referencing to the stream returned by split() function.
stream.on('data', (line) => console.log(`${line.toString('utf-8')}\n<---`))
stream.on('end', () => console.log('Done.'))
```

Example: Custom Transform Stream


```
{"id":991,"first_name":"Kristo","gender":"Male"}

{"id":992,"first_name":"Dawn","last_name":"Murrock","gender":"Female","ip_address":"216.229.117.43"
}

{"id":993,"first_name":"Guthrie","last_name":"Caudrey","email":"gcaudreyrk@e-recht24.de","gender":"
Male","ip_address":"206.1.106.191"}

{"id":994,"first_name":"Donaugh","last_name":"Marklin","email":"dmarklinrl@t.co","gender":"Male","i
p_address":"86.148.231.196"}

{"id":995,"first_name":"Ambrose","last_name":"Walklott","email":"awalklottrm@businesswire.com","gen
der":"Male","ip_address":"49.90.94.94"}
```

Example: Custom Transform Stream

```
const { Transform } = require('stream')
const transform = Transform({
 transform (chunk, encoding, callback) {
   ...
   return callback(null, <transformed-data>)
 }
})
```

```
const transform = Transform({
 transform (chunk, encoding, callback) {
   let obj
   // Parse JSON string
   try {
     obj = JSON.parse(chunk)
   } catch (err) {
     // We don't want to stop the transformation here due to a few errornous JSON.
     // instead emit the parsing error for consumers.
     this.emit('json_error', err)
     return callback()
   }
   ...
 }
})
```

```
const transform = Transform({
 transform (chunk, encoding, callback) {
   let obj
   // Parse JSON string
   try {
     obj = JSON.parse(chunk)
   } catch (err) {
     // We don't want to stop the transformation here due to a few errornous JSON.
     // instead emit the parsing error for consumers.
     this.emit('json_error', err)
     return callback()
   }
   // Ignore records without email address, emits an event
   if (!obj.email) {
     this.emit('empty_email', obj)
     return callback()
   }
   const data = {
     recipient: obj.email,
     content: `Hello, ${obj.first_name}.`
   }
   return callback(null, JSON.stringify(data) + '\n')
 }
})
```

```
transform.on('empty_email', (obj) => {
 // Count and print the records.
 counter += 1
 console.log(`No email address, ignoring: ${JSON.stringify(obj)}`)
})

// Read, gunzip, apply line break.
let stream = fs.createReadStream(sourcePath)
 .pipe(zlib.createGunzip())
 .pipe(split())
 .pipe(transform)
 .pipe(zlib.createGzip())
 .pipe(fs.createWriteStream(targetPath))
```

```
stream.on('error', (err) => {
 console.error(err)
 stream.end()
})

stream.on('finish', () => console.log(`Done writing output file. Records without email
                                       ${counter}.`))
```

Example: Using Stream for AWS S3 Object

```
// Uploading the file with Readable stream
    await s3.upload({
        Bucket: sourceBucket,
        Key: sourceKey,
        Body: fs.createReadStream(sourcePath)
    }).promise()
```

Convert JSON to CSV format into S3 object

```
    // Create a transform stream to convert JSON records to CSV format
    const transform = Transform({
        writableObjectMode: true,
        transform(obj, encoding, callback) {
            const csv = `"${obj.first_name}","${obj.last_name}", "${obj.gender}",
"${obj.email}"\n`
            return callback(null, csv)
        }
    })
```

```
// Create the transformation stream pipeline
const stream = s3.getObject({
        Bucket: sourceBucket,
        Key: sourceKey
    }).createReadStream()
    .pipe(zlib.createGunzip())
    .pipe(split(JSON.parse))
    .pipe(transform)
    .pipe(zlib.createGzip())

// Stream to target s3 object.
await s3.upload({
    Bucket: targetBucket,
    Key: targetKey,
    Body: stream
}).promise()
```

Example: Forking a stream

```
// Create the readable stream
const reader = fs.createReadStream(sourcePath)
 .pipe(zlib.createGunzip())
 // This time we parse JSON in split2 package
 .pipe(split(JSON.parse))

// Remember split() returns Transform stream that can also a Readable stream itself.
// Therefore `reader` now is a Readable stream that emits JSON objects as data.
```

```
function createFilter (gender) {
 // Create a transform stream according to gender.
 return Transform({
   // Indicate this stream will read data as object.
   writableObjectMode: true,
   transform (object, encoding, callback) {
     if (object.gender && object.gender === gender) {
       return callback(null, JSON.stringify(object) + '\n')
     }
     callback()
   }
 })
}
```

```
// Push data to output 1
const output1 = reader
 .pipe(createFilter('Male'))
 .pipe(zlib.createGzip())
 .pipe(fs.createWriteStream(outputPathMale))

// Also push data to output 2
const output2 = reader
 .pipe(createFilter('Female'))
 .pipe(zlib.createGzip())
 .pipe(fs.createWriteStream(outputPathFemale))

output1.on('finish', () => console.log('Done writing output 1.'))
output2.on('finish', () => console.log('Done writing output 2.'))
```

End

Helpful links:

- https://www.npmjs.com/package/split2
  - Break up a stream and reassemble it so that each line is a chunk.
- https://www.npmjs.com/package/through2
  - A tiny wrapper around Node streams.