Universidad de La Laguna. Escuela Técnica Superior de Ingeniería Informática
Tercero del Grado de Informática
DESARROLLO DE SISTEMAS INFORMÁTICOS: CONVOCATORIA DE JULIO
03/07/2017 4 páginas

Nombre: _____
Alu: _____ GitHub Id: _____

Descripción del Problema Te dan una lista `path` de caminos a ficheros en disco. Cada fichero contiene un número y debes dar como resultado una string que contenga los números en los ficheros **en el orden dado en la lista** `path`. Para hacer el problema mas asíncrono se debe esperar en cada lectura un número aleatorio entre 0 y 10 de milisegundos antes de añadir el número leído a la string (véase ejemplo mas abajo). Por supuesto, no se permite en este problema hacer lectura síncrona. Se debe usar `readFile`.

Una Solución Errónea Este es un ejemplo del tipo de solución que puedes dar. Esta solución es errónea:

```
var fs = require('fs');
var paths = [
             'first-file',  // contains 10
             'second-file',  // contains 7
             'third-file'  // contains 5
             ];
var FinalResult = "";

paths.forEach(function(path) {
    fs.readFile(path, 'utf8', function(err, data) {
        // here we wait for random time
        setTimeout(function() {
            FinalResult += data.replace(/^\s*|\s*$/g,'')+ " "; // trim spaces
            console.log(FinalResult);
        }, Math.floor(Math.random() * 10));

    });
}, () => console.log("FinalResult = "+FinalResult));
```

Cuando se ejecuta, el orden en que ocurren las tareas es impredecible:

```
[~/javascript/learning/promises/async-examples/part1(master)]$ node sol12wrong.js
10
10 7
10 7 5
[~/javascript/learning/promises/async-examples/part1(master)]$ node sol12wrong.js
5
5 7
5 7 10
[~/javascript/learning/promises/async-examples/part1(master)]$ node sol12wrong.js
10
10 5
```

Este programa posiblemente imprime los números en un orden incorrecto

pipe Se pide que escribas un método `pipe` para los objetos `Array` que recibe como primer argumento la funcion `asyncTask` conteniendo la tarea asíncrona a realizar sobre cada objeto del array y como segundo argumento una `callback` que será llamada cuando todas las iteraciones asíncronas hayan terminado. Las tareas son ejecutadas en el orden establecido por el array:

```
Array.prototype.pipe = function(asyncTask, callback) {
    ....
}
```

La tarea `asyncTask` recibe como primer argumento el elemento del array a procesar y como segundo argumento una callback a la que deberá llamar para indicar que ha finalizado su tarea:

```
asynctask = function(item, nextTask) {
  ... // do your async task
  nextTask();
}
```

Programa de prueba `Array.prototype.pipe` debe estar escrita de manera que se pueda dar una solución al problema propuesto usando dicha función. Una vez la tengas escrita, y guardada en un módulo `pipe`, este programa debería funcionar correctamente:

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

```
var paths = [
             'first-file',  // contains 10
             'second-file',  // contains 7
             'third-file'  // contains 5
             ];
var FinalResult = "";

var readFilesInOrder = (path, nextTask) => {
  fs.readFile(path, 'utf8', function(err, data) {
      // here we wait for random time
      setTimeout(function() {
          FinalResult += data + " ";
          nextTask();
      }, Math.floor(Math.random() * 10));

  });
};

paths.pipe(readFilesInOrder, () => { console.log(FinalResult); });
```

Cuando se ejecuta los números salen en orden:

```
[~/javascript/async-examples/part1(master)]$ node sol12.js
10
 7
 5
```

Requisitos

1. **Escribe un módulo npm que extiende la clase `Array` con el método `pipe`**:

   ```
   tasksToDo.pipe(asyncTask, doItAfter);
   ```

   que ejecuta las funciones asíncronas `asyncTask` sobre los objetos en el array `tasksToDo` en el orden determinado por el array; esto es, hace primero `asyncTask(tasksToDo[0])`, cuando este termina `asyncTasks(tasksToDo[1])`, etc. y ejecuta `doItAfter` cuando todas las `asyncTask` han terminado.

2. **Explica como es el proceso para publicar lo hecho como módulo en npm**.

3. **Explica como ejecutar las pruebas usando Mocha, y Chai-Should**. Aqui dejamos un esqueleto del que puedes partir:

```
var should = require('chai').should();
var ins = require("util").inspect;

describe('testing something', function() {
  it('does something', function() {
    let input = "...";
    let r = ....; // call the method to test
    let expected = ... ; // what is expected
    r.should.deep.equal(expected);
  });
});
```