UNIVERSITÀ DI TORINO

Exercise C Working with Promises

Prof. Fabio Ciravegna
Dipartimento di Informatica
Università di Torino
fabio.ciravegna@unito.it

UNIVERSITÀ DI TORINO

Promises

- In this exercise we will work with promises
- In particular we will convert an express route based on callbacks
  - into an equivalent route using promises
- We will first see this in action with an example and then you will be asked to do a similar exercise

Writing two files

- We will use an npm library called fs which is used to access files
  - All functions accessing files are asynchronous operations
- The code provided (Week 2.c Lab Class - Promises Introduction) shows an eps file with two buttons
  - each of them fetches a route in routes/index.js
  - the first route writes the two files using callbacks
  - the second route does the same operations using promises
- Note:
  - we will use the npm module fs in its base form
    - however fs has a version using promises - so it is suggested that after today you use the versions with promises directly rather than the base implementation

UNIVERSITÀ DI TORINO

With callbacks

```
/**
 * this route writes two files iin sequence using callbacks
 */
router.get('/get_photos', function (req, res, next) {
    const data = "console.log('Hello World')";
    fs.writeFile('file1.js', data, (error) => {
        if (error) {
            console.error(err);
            res.writeHead(500, {'Content-Type': 'text/plain'});
            res.end('error in writing files' + err);
        }
        console.log('The file2.js has been saved!');
        fs.writeFile('file2.js', data, (error) => {
            if (error) {
                console.error(err);
                res.writeHead(500, {'Content-Type': 'text/plain'});
                res.end('error in writing files' + err);
            }
            console.log('The file1.js has been saved!');
            res.writeHead(200, {'Content-Type': 'text/plain'});
            res.end('both files were written');
        });
    });
});
```

UNIVERSITÀ DI TORINO

With Promises

```
let writeFilePromise= function (data, path) {
    return new Promise((resolve, reject) => {
        fs.writeFile('file2.js', data, (error) => {
            if (error) reject();
            else resolve();
        })
    });
}
router.get('/get_photos_promises', function (req, res, next) {
    const data = "console.log('Hello World')";
    writeFilePromise(data, './public/images/image1.png')
        .then(() => writeFilePromise(data, './public/images/image2.pn
        .then(() => {
            res.writeHead(200, {'Content-Type': 'text/plain'});
            res.end('both files were written');
        })
        .catch(err => {
            console.error(err);
            res.writeHead(500, {'Content-Type': 'text/plain'});
            res.end('error in writing files' + err);
        })
});
```

UNIVERSITÀ DI TORINO

Exercise

- Do a similar task with another function from the fs module: fs.access
  - which checks if a file exists although formally:
    The **fs.access() method** is used to test the permissions of a given file or directory.
- You are given a version of a programme that checks if three images exist
  - the images are under /public/images/
  - the code uses callbacks
- Replace the code using promises

UNIVERSITÀ DI TORINO

Node.js fs.access() Method

Read Discuss Courses

The **fs.access() method** is used to test the permissions of a given file or directory. The permissions to be checked can be specified as a parameter using file access constants. It is also possible to check multiple file permissions by using the bitwise OR operator to create a mask with more than one file constant.
Note: It is not recommended to use the fs.access() method to check for the accessibility of a file before calling fs.open(), fs.readFile() or fs.writeFile(), because it introduces a race condition since the file state may be changed by other processes after the test.

Syntax:

```
fs.access( path, mode, callback )
```

Parameters: This method accepts three parameters as mentioned above and described below:

- **path:** It is a String, Buffer or URL that denotes the path of the file or directory for which the permission has to be tested.
- **mode:** It is an integer value that denotes the permission to be tested for. The logical OR operator can be used to separate multiple permission. It can have the values fs.constants.F_OK, fs.constants.R_OK, fs.constants.W_OK and fs.constants.X_OK. It is an optional parameter. The default value is fs.constants.F_OK.
- **callback:** It is a function that would be called when the method is executed.
  - **err:** It is an error that would be thrown if the method fails.

https://www.geeksforgeeks.org/node-js-fs-access-method/

UNIVERSITÀ DI TORINO

javascript

```javascript
// Node.js program to demonstrate the
// fs.access() method

// Import the filesystem module
const fs = require('fs');

// Allowing only read permission
console.log("Giving only read permission to the user");
fs.chmodSync("example_file.txt", fs.constants.S_IRUSR);

// Test the read permission
fs.access('example_file.txt', fs.constants.R_OK, (err) => {
  console.log('\n> Checking Permission for reading the file');
  if (err)
    console.error('No Read access');
  else
    console.log('File can be read');
});

// Test both the read and write permissions
fs.access('example_file.txt', fs.constants.R_OK
                  | fs.constants.W_OK, (err) => {
  console.log('\n> Checking Permission for reading"
                          + " and writing to file');
  if (err)
    console.error('No Read and Write access');
  else
    console.log('File can be read and written');
});
```

UNIVERSITÀ DI TORINO

Questions?

© Prof. Fabio Cravegna, Università di Torino