Multi threading

Thibaud Cruzille - Rémy Moulaire - Rémy Viniacourt

Sommaire

- L'objectif du projet
- Les librairies
- Les threads
- Les fonctions
- Quelques exemples de code
- Les promesses
- Démonstration

L’objectif du projet

- Scanner le contenu d’un dossier
- Détecter les fichiers avec un format spécifique (.jpg)
- Vérifier si ces images ont des métadonnées
- Redimensionner l’image
- Stocker les descriptions dans une base de données

- Permettre à l’utilisateur de rechercher un mot clé

Thibaud Cruzille - Rémy Moulaire - Rémy Viniacourt

Les différentes librairies

Path

Sharp

Readline

Mongoose

Threading

Un main thread :

Il fera l'interaction avec l'utilisateur : recherche par key word.

Des workers pour les dossiers :

Lancent l'analyse des dossiers toutes les 5 secondes. Réalise les différents traitements nécessaire si de nouvelles images ont été importées.

Les fonctions

- Process Folder :
  - scan du dossier (récursif)
  - créer l'arborescence dans le dossier de sortie (resized)
  - lance fonction traitement image
- Traitement image :
  - parse le fichier metadata
  - insère les données dans MongoDB (metadata + path de l'image resized)
  - redimensionne l'image (envoi dans dossier resized)
- Cleandirectory : supprime le contenu du dossier spécifié (récursif)
- Ask : saisie utilisateur
- Workers : appelle process folder et exécute ensuite cleandirectory

Des exemples de code

MongoDB -› Mongoose

```
// Inclusion de Mongoose
mongoose.set('useNewUrlParser', true);
mongoose.set('useUnifiedTopology', true);
mongoose.connect('mongodb://localhost/key_word', { useNewUrlParser: true });

var keyWordSchema = new mongoose.Schema({
    Filename : String,
    Keyword : String
});
```

Thibaud Cruzille - Rémy Moulaire - Rémy Viniacourt

Appel à MongoDB -› Mongoose

```
var monCommentaire = new keyWordModel({ Filename : image_output_path });
monCommentaire.Keyword = lines[i];
monCommentaire.save(function (err) {
    if (err) { throw err; }
    console.log('Commentaire ajouté avec succès !');
});
```

```
var query = keyWordModel.distinct('Filename', { "Keyword" : clef } );
query.exec(function (err, comms) {
    if (err) { throw err; }
    // On va parcourir le résultat et les afficher joliment
    var comm;
    for (var i = 0, l = comms.length; i < l; i++) {
      comm = comms[i];
      console.log('\n');
      console.log('------------------------------');
      console.log('Filename : ' + comm);
      console.log('------------------------------');
    }
  });
```

Les librairie fs et readline

Lire des fichiers

Supprimer des fichiers

Créer un dossier

Supprimer un dossier

```
var files=fs.readdirSync(startPath);
for(const file of files)
{
```

```
const r1 = readline.createInterface({

    input: process.stdin,
    output: process.stdout
});
```

```
/* Fonction qui va permettre de demander à l'utilisateur de saisir un key word pour la recherche */
function ask(questionText) {
    return new Promise((resolve, reject) => {
      r1.question(questionText, resolve);
    });
}
```

Des promesses ....

Une promesse est un objet (`Promise`) qui représente la complétion ou l'échec d'une opération asynchrone

```
const csv_promise = new Promise(function(resolve, reject){ //la csv promise est une promise qui traite le fichier csv
```

```
// resize promise est une promise fournie par sharp qui redimensionne l'image
const resized_promise = sharp(image_path).resize(50, 50).png().toFile(image_output_path).then(() => console.log(image_path + ' processed'));
// on veut attendre que les deux promise ( csv et sharp ) soient completes donc on renvoie une promise composite
return Promise.all([csv_promise, resized_promise]);
```

Chaînages de promesses

- Les *callbacks* ajoutés grâce à `then` seront appelés, y compris après le succès ou l'échec de l'opération asynchrone qui la précède.
- Plusieurs *callbacks* peuvent être ajoutés en appelant `then` plusieurs fois, ils seront alors exécutés l'un après l'autre selon l'ordre dans lequel ils ont été insérés.

```
function FromDirs(){
    Process_folder('./image',folder_out, '.jpg').then( () => clean_directory('./image')) ;
    Process_folder('./image1',folder_out, '.jpg').then( () => clean_directory('./image1')) ;
    Process_folder('./image2',folder_out, '.jpg').then( () => clean_directory('./image2')) ;
    setTimeout(FromDirs, 5000); // appel des 3 dossiers toutes les secondes task va appeler fromdir 3 fois
}
```

Et une démonstration

Lien vers github :
https://github.com/remyvin/tse_de3_multithreading