

10. Async Programming

Jean-Luc Falcone

Août 2019

Style Direct

```
function rng() {  
  return Math.random() * 200 - 100;  
}  
function plus10( x ) {  
  return x+10;  
}  
function sqrtAbs( x ) {  
  return Math.sqrt( Math.abs );  
}  
let x = rng();  
let y = plus10(x);  
let z = sqrtAbs(y);  
console.log(z);
```

Style par passage de continuation (CPS)

```
function rng(RETURN) {  
  RETURN( Math.random() * 200 - 100 );  
}  
  
function plus10( x, RETURN ) {  
  RETURN(x+10);  
}  
  
function sqrtAbs( x, RETURN ) {  
  RETURN( Math.sqrt( Math.abs(x) ) );  
}  
  
rng(function(x) {  
  plus10( x, function(y) {  
    sqrtAbs( y, console.log );  
  });  
});
```

Comparaison

Direct

```
let x = rng();  
let y = plus10(x);  
let z = sqrtAbs(y);  
  
console.log(z);
```

Continuations

```
rng( x => {  
  plus10( x, y => {  
    sqrtAbs( y, console.log );  
  });  
});
```

Modularisation

Direct

```
function foo() {  
  let x = rng();  
  let y = plus10(x);  
  return sqrtAbs(y);  
}
```

```
console.log(foo());
```

Continuations

```
function foo(RETURN) {  
  rng( x => {  
    plus10( x, y => {  
      sqrtAbs( y, RETURN );  
    });  
  });  
};
```

```
foo( console.log );
```

Synchrone, style direct

```
try {  
  fs.writeFileSync("foo.txt", foo, "utf8");  
  fs.writeFileSync("goo.txt", goo, "utf8");  
  fs.writeFileSync("hoo.txt", hoo, "utf8");  
  fs.writeFileSync("ioo.txt", ioo, "utf8");  
  fs.writeFileSync("joo.txt", joo, "utf8");  
} catch(e) {  
  errorHandler(e);  
}
```

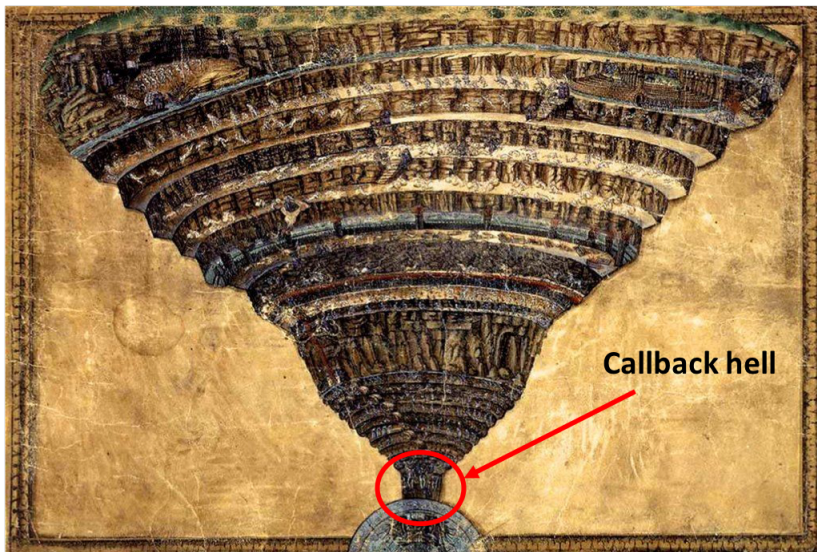
Asynchrone, CPS

```
fs.writeFile( "foo.txt", foo, "utf8", (err) => {  
  if( err === undefined ) {  
    fs.writeFile("goo.txt", goo, "utf8", (err) => {  
      if( err === undefined ) {  
        fs.writeFile("hoo.txt", hoo, "utf8", (err) => {  
          if( err === undefined ) {  
            fs.writeFile("ioo.txt", ioo, "utf8", (err) => {  
              if( err === undefined ) {  
                fs.writeFile("joo.txt", joo, "utf8", (err) => {  
                  if( err !== undefined ) {  
                    errorHandler(e);  
                  }  
                });  
              } else errorHandler(err);  
            });  
          } else errorHandler(err);  
        });  
      } else errorHandler(err);  
    });  
  }  
});
```



icompile.eladkarako.com

Callback Hell



Synchrone, style direct... et moche !

```
try {  
  fs.writeFileSync("foo.txt", foo, "utf8");  
  fs.writeFileSync("goo.txt", goo, "utf8");  
  fs.writeFileSync("hoo.txt", hoo, "utf8");  
  fs.writeFileSync("ioo.txt", ioo, "utf8");  
  fs.writeFileSync("joo.txt", joo, "utf8");  
} catch(e) {  
  errorHandler(e);  
}
```

Boucle synchrone, direct

```
function writeAll(names,data) {  
  try {  
    for( let i = 0; i<files.length; i++ ) {  
      fs.writeFileSync(names[i], data[i], "utf8");  
    }  
  } catch(e) {  
    errorHandler(e);  
  }  
}
```

Boucle asynchrone, CPS

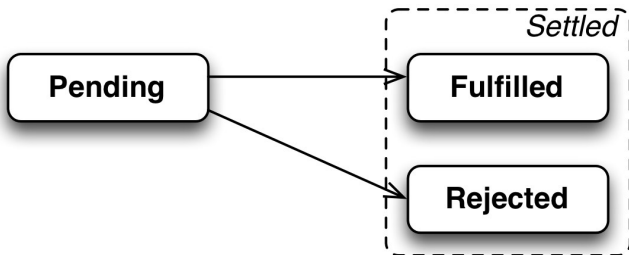
```
function writeAll(names,data,i,errorHandler) {  
  if( index < names.length ) {  
    fs.writeFile(names[i], data[i], function(err) {  
      if( err !== undefined ) errorHandler(err);  
      else  
        writeAll( names,data,i+1,errorHandler );  
    });  
  }  
}
```

Boucle asynchrone, CPS, et modularisé

```
function writeAll(names,data,i,done,errorHandler) {  
  if( index < names.length ) {  
    fs.writeFile(names[i], data[i], function(err) {  
      if( err !== undefined ) errorHandler(err);  
      else  
        writeAll( names,data,i+1,done,errorHandler );  
    });  
  } else done();  
}
```

Promises (ES6)

- Une Promise (promesse) est un objet qui encapsule une exécution asynchrone.
- Existe dans d'autres langages comme Java, C++, etc. (Future)
- Elle permet de chaîner des appels asynchrones plus aisément.



Exemple sans promesses

Les fonctions foo, hoo et goo sont asynchrones:

```
foo(10, (x) => {  
  hoo( x, (y) => {  
    goo( y, console.log );  
  });  
});
```

Exemple avec promesse

Les fonctions `foo`, `hoo` et `goo` sont asynchrones et retournent des promesses:

```
foo(10)
  .then( hoo )
  .then( goo )
  .then( console.log );
```


Chaînage

- La fonction `then` qui permet de chaîner les callbacks.
- Invoqués lorsque la promesse est remplie (succès).
- Retourne de nouvelles promesses

Chaînage: Exemple

```
compute(). //Retourne une promesse
  .then( (x) => {
    console.log("Le resultat est: " + x );
    return x*x;
  })
  .then( (x) => {
    console.log("Le carré du resultat est: " + x );
  });
```

Chaînage: Exemple (2)

//X est aussi une promesse

```
let x = compute().  
      .then(Math.abs);
```

```
x.then( console.log );
```

Exemple avec erreur, sans promesses

```
foo(10, (x,error) => {  
  if( error !== undefined ) {  
    hoo(x, (y,error) => {  
      if( error !== undefined ) {  
        console.log("Result: " + y );  
      } else {  
        console.error("Error: " + error);  
      }  
    });  
  } else {  
    console.error("Error: " + error);  
  }  
});
```

Exemple avec erreur, avec promesses

La fonction `catch` fonctionne comme `then` mais est invoquée en cas d'erreurs...

```
foo(10)
  .then(hoo)
  .then( x => console.log("Result: " + x );
  .catch( error => console.error("Error: " + error) );
```

Garanties

- Chaque callback n'est invoqué qu'une fois l'itération de la boucle principale terminée.
- Chaque callback passé avec `then` est invoqué sera appelé, même si le résultat est connu (promesse remplie).
- Les callbacks passés avec `then` sont exécuté dans l'ordre d'insertion.

"Promise-ifier" un code existant

```
function readData( name ) {  
  return new Promise( (resolve,reject) => {  
    fs.readFile( name, "utf8", (err,data) => {  
      if( err !== undefined ) reject(err);  
      else resolve(data);  
    });  
  });  
}
```

```
function writeData( name, data ) {  
  return new Promise( (resolve,reject) => {  
    fs.writeFile( name, "utf8", (err) => {  
      if( err !== undefined ) reject(err);  
      else resolve();  
    });  
  });  
}
```

"Promesse-ifier" un code existant

```
function copyData( src, dest ) {  
  return readData( src )  
    .then( (data) => writeData( dest, data ) );  
}
```

```
copyData( "machin.txt", "chose.txt" )  
  .catch( console.error );
```


Await/async (ES7)

```
async function copyData( src, dest ) {  
  let data = await readData( src );  
  return writeData( dest, data );  
}
```

```
copyData( "machin.txt", "chose.txt" )  
  .catch( console.error );
```

Quelques liens

- [Promise API \(MDN\)](#)
- [Using Promises \(MDN\)](#)
- [Async/Await \(MDN \)](#)
- [Async/Await in loops \(blog post\)](#)