Async & Await

Rujuan Xing

Maharishi International University - Fairfield, Iowa

Async/await

- It’s a special syntax to work with promises in a more comfortable fashion
- The `async` keyword: when you put `async` keyword in front of a function declaration, it turns the function into an `async` function.
- The `await` keyword: `await` only works inside `async` functions. `await` can be put in front of any `async` promise-based function to pause your code on that line until the `promise` fulfills, then return the resulting `value`.

Async functions

- `async` can be placed before a function. An `async` function always returns a `promise`:
  - When no return statement defined, or return without a value. It turns a resolving a promise equivalent to `return Promise.Resolve()`
  - When a return statement is defined with a value, it will return a resolving promise with the given return `value`, equivalent to `return Promise.Resolve(value)`
  - When an error is thrown, a rejected promised will be returned with the thrown error, equivalent to `return Promise.Reject(error)`

```
console.log('start');
async function f() {
    return 1;
}

f().then(console.log);
console.log('end');
```

Await

- The keyword `await` makes JavaScript wait until that `promise` settles and returns its result.
- `await` literally suspends the function execution until the `promise` settles, and then resumes it with the `promise` result. That doesn't cost any CPU resources, because the JavaScript engine can do other jobs in the meantime: execute other scripts, handle events, etc.
- It's just a more elegant syntax of getting the `promise` result than `promise.then`.

```
console.log('start');
async function foo() {
    return 'done!';
}
async function bar() {
    console.log('inside bar - start');
    let result = await foo();
    console.log(result); // "done!"
    console.log('inside bar - end');
}
bar();
console.log('end');
```

Await (cont.)

1. Can't use `await` in regular functions. If we try to use `await` in a non-async function, there would be a syntax error:

```
async function foo() {
    return 'done!';
}

function bar() {
    let result = await foo(); // Syntax error
    console.log(result);
}
bar();
```

2. `await` won't work in the top-level code

```
// syntax error in top-level code
async function baz() {
    return 'baz...';
}

let result = await baz(); //Syntax Error
console.log(result);
```

Error handling in Synchronous function and Asynchronous function

- In asynchronous function, the regular `try…catch` is not able to catch the error.
  - For example: as the right side `thisThrows()` is `async`, when we call it, it dispatches a `promise`, the code doesn't wait, so the `finally` block is executed first and then the promise executes, which is then rejects.

```
//synchronous function

function thisThrows() {
    throw new Error("Thrown from thisThrows()");
}

try {
    thisThrows();
} catch (e) {
    console.error(e);
} finally {
    console.log('We do cleanup here');
}
```

```
//asynchronous function

async function thisThrows() {
    throw new Error("Thrown from thisThrows()");
}

try {
    thisThrows();
} catch (e) {
    console.error(e);
} finally {
    console.log('We do cleanup here');
}
```

Error Handling in Async functions

1. Use `await` inside async function

2. Chain async function call with a `.catch()` call.

```
async function thisThrows() {
    throw new Error("Thrown from thisThrows()");
}

async function run() {
    try {
        await thisThrows();
    } catch (e) {
        console.log('Caught the error....');
        console.error(e);
    } finally {
        console.log('We do cleanup here');
    }
}

run();
```

```
async function thisThrows() {
    throw new Error("Thrown from thisThrows()");
}

thisThrows()
    .catch(console.error)
    .finally(() => console.log('We do cleanup here'));
```

How Async & Await solve callback hell issue?

```
function multiplyBy10Async(num, callback) {
    setTimeout(callback, 1000, num * 10);
}

function withCallback() {
    multiplyBy10Async(5, (num1) => {
        multiplyBy10Async(num1, (num2) => {
            console.log(num2);
        });
    });
}

withCallback();
```

```
function multiplyBy10Async(num, callback) {
    setTimeout(callback, 1000, num * 10);
}

function multiplyBy10Promise(num) {
    return new Promise(function(resolve, reject) {
        multiplyBy10Async(num, (result) => resolve(result));
    });
}

async function withAsyncAndAwait() {
    let res1 = await multiplyBy10Promise(5);
    let res2 = await multiplyBy10Promise(res1);
    console.log(res2);
}

withAsyncAndAwait();
```