Async Javascript

Synchronous

Note:Javascript is a single threaded, sychronous language.

There is a example for synchronous:

```
// 1.When we make a soup, there's a recipe for that:
// 2.Chop onion
// 3.Chop carrots
// 4.Boil water 10 min
// 5.add onion boil for 5min
// 6.add carrots boil for 5min
console.log('Chop onion');
console.log('Chop carrot');
//Boil water 10 min
boilingWater(100000);
//add carrots boil for 5 min
boilingWater(50000);
//add carrots boil for 5 min
boilingWater(50000);
/*But in fact we found that when the water is boiling we have nothing to do, so
why not we start boiling water when we chop the onion, so we arrange the order:*/
// Boil water 10 min
// Chop onion
// add onion boil for 5min
// Chop carrots
// add carrots boil for 5min
/*But the fact is even when we change the order of the function we still need to
wait the water has boiled,then we could do the other things in javascript, and
this is the sychronous means*/
function boilingWater(time) {
  console.log('boiling...');
  for (let i = 0; i < time; i++) {
    console.log('still not done');
  }
  console.log('done');
}
```

Asynchronous

In fact when we use asynchronous we do not change the principle of javascript, javascript is still a single threaded language. Because the way we achieve the asynchronous in javascript, we depend on the methods in browser instead the methods in JavaScript, because the it's single threaded, but when we use browser's methods, it becomes different, and these methods include such as setTimeout, Fetch Data, Get Geolocation etc. When we use a browser method, like setTimeout:

```
  console.log("I am busy");
}
console.log("I have time now");
funcion boilingWater() {
  console.log('boiling');
  setTimeout(function () {
    console.log('done');
  }, 1000);
}
/*In this code we will find "console.log('done');" won't work until
"console.log("I have time now");" has done. Firstly we found that after we use
timeout, even it doesn't completed we could still enter "for loop", and it is the
asynchronous means.Besides we found the timeout wont work until the "for loop" has
done. Why? Because "for loop" is a javascript behavior, but setTimeout is a
browser behavior
*/
```

But in fact if we use this way to solve the problem above we will be suck to the "Callback Hell" like this:

```
boilWater();
console.log('Chop carrot');
function boilWater() {
  console.log('boiling...');
  setTimeout(() => {
    console.log('done');
    console.log('add carrots');
    setTimeout(() => {
      console.log('carrots done');
      console.log('add onions');
      setTimeout(() => {
        console.log('onion done');
      }, 500);
    }, 500);
  }, 1000);
}
```

```
result:
boiling...
test.js:114 Chop carrot
test.js:118 done
test.js:119 add carrots
test.js:121 carrots done
test.js:122 add onions
test.js:124 onion done
```

Promises

The Promises is an object that has three status: Pending,Resolved and Rejected.The Pending status is the default status. We could use resolve and reject to control the return:

```
const promise = new Promise((resolve, reject) => {
  let value = true;
  if (true) {
    resolve(() => {
      console.log('The value is true');
    });
  } else {
    reject('The value is false!');
  }
});
promise
  .then((resolved) => {
    resolved();
  })
  .catch((rejected) => {
    console.log(rejected);
  });
const promiseObject = new Promise((resolve, reject) => {
  let value = true;
  if (value) {
    resolve('The value is true!');
    /*Resolve and Reject functions can return any thing you want*/
  } else {
    reject('The value is false!');
  }
});
promiseObject
  .then((resolved) => {
    /*We use then to operate resolved one and catch to return the rejected one*/
    console.log(resolved);
  })
  .catch((rejected) => {
    console.log(rejected);
  });
```

For deeper understand of the promise, there's an example for you to understand when to use resolve and reject :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      body {
        font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,
```

```
          Oxygen, Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
        text-align: center;
      }
      h1 {
        letter-spacing: 2px;
        text-align: center;
      }
      img {
        width: 100%;
        height: 400px;
        object-fit: cover;
      }
    </style>
  </head>
  <body>
    <h1>Asynchronous Javascript</h1>
    <h1 class="one">hello world</h1>
    <h1 class="two">hello people</h1>
    <h1 class="three">hello Javascript</h1>
    <div class="img-container">
      <!-- <img src="https://source.unsplash.com/random" alt="random image" /> -->
    </div>
    <button class="btn">click me</button>
    <script src="./app.js"></script>
  </body>
</html>
```

```
// callbacks, promises, async/await
const heading1 = document.querySelector('.one');
const heading2 = document.querySelector('.two');
const heading3 = document.querySelector('.three');
const btn = document.querySelector('.btn');
const container = document.querySelector('.img-container');
const url = 'https://source.unsplash.com/random';
btn.addEventListener('click', () => {
  loadImage(url)
    .then((taco) => container.appendChild(taco))
    .catch((err) => console.log(err));
});

function loadImage(url) {
  return new Promise((resolve, reject) => {
    let img = new Image();
    img.addEventListener('load', () => {
      resolve(img);
    });
    img.addEventListener('error', () => {
      reject(new Error(`Failed to load image from the source : ${url}`));
    });
    img.src = url;
```

```
  });
}
```

And here's another case to understand th actual use:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      body {
        font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,
          Oxygen, Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
        text-align: center;
      }
      h1 {
        letter-spacing: 2px;
        text-align: center;
      }
      img {
        width: 100%;
        height: 400px;
        object-fit: cover;
      }
    </style>
  </head>
  <body>
    <h1>Asynchronous Javascript</h1>
    <h1 class="one">hello world</h1>
    <h1 class="two">hello people</h1>
    <h1 class="three">hello Javascript</h1>
    <div class="img-container">
      <!-- <img src="https://source.unsplash.com/random" alt="random image" /> -->
    </div>
    <button class="btn">click me</button>
    <script src="./app.js"></script>
  </body>
</html>
```

```
// callbacks, promises, async/await
// what if no resolve, one is rejected
const heading1 = document.querySelector('.one');
const heading2 = document.querySelector('.four');
const heading3 = document.querySelector('.three');
const btn = document.querySelector('.btn');
btn.addEventListener('click', () => {
  addColor(1000, heading1, 'red')
```

```
    .then(() => addColor(2000, heading2, 'green'))
    .then(() => addColor(1000, heading3, 'blue'))
    .catch((err) => console.log(err));
});

function addColor(time, element, color) {
  return new Promise((resolve, reject) => {
    // Create a promise
    if (element) {
      //When this element exists , we run the later code
      setTimeout(() => {
        //We se a timer to edit the style of the element
        element.style.color = color;
        resolve(); /*Be careful, this method need the parameter only when you want
this promise return, but in this case, we need just change the style of the
element, but the style has already been changed we do not need return the value,
so this is case is unlike the last one*/
      }, time);
    } else {
      reject(new Error(`There is no such element ${element}`));
    }
  });
}
```

Async/Await

Use this keyword we could make our code more readable:

```
//Two ways to create async function
// async function someFunction (){
//   await
// }
// const otherFunction = async() =>{
//   await
// }

const heading1 = document.querySelector('.one');
const heading2 = document.querySelector('.two');
const heading3 = document.querySelector('.three');
const btn = document.querySelector('.btn');
btn.addEventListener('click', async () => {
  const result = await displayColor();
  console.log(result);
});

async function displayColor() {
  try {
    await addColor(1000, heading1, 'red');
    await addColor(1000, heading2, 'green');
    await addColor(1000, heading3, 'blue');
  } catch (error) {
```

```
    console.log(error);
  }
  return 'hello';
}

function addColor(time, element, color) {
  return new Promise((resolve, reject) => {
    if (element) {
      setTimeout(() => {
        element.style.color = color;
        resolve();
      }, time);
    } else {
      reject(new Error(`There is no such element ${element}`));
    }
  });
}
```