04

Javascript -3

모듈, 비동기 처리, 클래스

모듈

모듈

- 함수, 변수, 클래스 등으로 이루어진 파일이나 라이브러리
- CommonJS - Node.js 에서 사용하는 모듈 시스템
- ES6 Module - ES6 버전에서 정의한 모듈 시스템. Vue.js 등에서 사용

- 웹 브라우저에서는 모듈을 지원하지 않는 경우가 대부분이므로 모듈은 node 환경에서 구현한다.

CommonJS

- 내보내는 쪽
  - module.exports 로 정의된 객체, 함수 등이 외부로 전달된다.
- 사용하는 쪽
  - require(‘모듈’) 에서 반환된 객체, 함수 등을 변수에 저장하여 사용한다.

ES6 Module

- 내보내는 쪽
  - export 키워드를 이용해 외부에 전달할 항목을 지정
  - 하나만 내보낼 때는 export default 키워드 사용
- 사용하는 쪽
  - import 이름 from 파일
  - destructuring을 이용해 일부만 부르기 가능
    - import { 항목 } from 모듈

CommonJS

작업 폴더 및 파일 준비

- CommonJS 방식의 모듈을 사용하기 위해 작업용 폴더를 하나 만들고
- 다음과 같이 JS 파일들을 생성한다. (빈 파일 생성)
  - main.js
  - each.js : 항목을 하나씩 추가
  - obj.js : 객체를 이용해 한번에 추가

obj.js

- 필요한 값, 함수를 가진 객체를 만든 다음 module.exports 에 해당 객체를 할당한다.

```
1  const data = {
2      user:'abcd',
3      role:'admin',
4      sayHello() {
5          console.log('Hello!!!');
6      }
7  };
8
9  // exports 객체를 내가 만든 객체로 한번에 할당
10 module.exports = data;
```

each.js

- 함수나 변수를 module.exports 객체에 필요한 만큼 각각 추가한다.

```
1  // 항목을 하나씩 exports 객체에 추가하는 방법
2  module.exports.message = 'Util module';
3
4  module.exports.hello = (user)=>{
5      console.log(`Hello ${user}`);
6  };
7
8  module.exports.bye = ()=>{
9      console.log('Bye');
10 };
```

main.js

- require 문구를 이용해 필요한 모듈을 불러와 객체에 저장한다.

```
1  const data = require('./obj');
2  const util = require('./each');
3
4  console.log(data);
5  data.sayHello();
6
7  console.log(util.message)
8  util.hello(data.user);
9  util.bye();
```

ES6 Module

each.js

- each.js를 ES6 용으로 모듈을 만들기 위해서는 다음과 같이 코드를 작성한다.

```
1  // 항목을 하나씩 export 하는 방법
2  const message = 'Util module';
3
4  const hello = (user)=>{
5      console.log(`Hello ${user}`);
6  };
7
8  const bye = ()=>{
9      console.log('Bye');
10 };
11
12 export {
13     message,
14     hello,
15     bye
16 }
```

obj.js

- 하나의 객체로 반환하기 위해서는 다음과 같이 수정할 수 있다.

```
export default {
    user:'abcd',
    role:'admin',
    sayHello() {
        console.log('Hello!!!');
    }
};
```

package.json

- node 모듈은 기본 값으로 CommonJS 방식으로 동작한다.
- ES6 모듈을 사용하려면 소스코드가 있는 폴더에 package.json 파일을 추가하고 다음과 같이 모듈 타입을 지정해줘야 한다.

```
1  {
2      "type":"module"
3  }
```

main.js

- ES6 모듈로 고친다.

```
1   //const data = require('./obj');
2   //const util = require('./each');
3
4   import obj from './obj.js'; // 이름은 자유롭게 지을 수 있다.
5   import {message, hello, bye} from './each.js'; // 이름을 맞춘다.
6
7   console.log(obj);
8   obj.sayHello();
9
10  console.log(message);
11  hello(obj.user);
12  bye();
```

비동기 처리

동기 - 비동기

- 동기 함수: 함수를 호출하면 그 함수가 return 하기 전까지 다음 줄의 코드가 실행되지 않는다.

```
1  function sigma(start, end){
2      let sum=0;
3      for(let i=start; i < end; i++){
4          sum += i;
5      }
6      return sum;
7  }
8
9  const start=1;
10 const end = 10000;
11
12 // sigma 함수가 반환할 때까지 result의 할당은 이루어지지 않으며
13 const result = sigma(start, end);
14 // console.log 역시 실행되지 않는다.
15 console.log(result);
```

동기 - 비동기

- 비동기 함수: 함수를 호출하면 그 함수는 일단 바로 return 하고 실행 결과는 나중에 알려준다.
  - 콜백: 실행 결과를 처리할 함수를 파라미터로 넘기는 방법.
  - 여러개의 비동기 함수가 순차적으로 처리되어야 할 때, 콜백은 코드 깊이가 너무 깊어지는 단점이 있다.

Promise

- 비동기적인 동작에서 callback의 사용을 줄이는 방법
- Promise는 3가지 상태를 가지는 객체이다.
  - Pending: 객체가 생성되었지만 동작은 완료되지 않은 상태. 파라미터로 resolve, reject 함수를 받는다.
  - Fulfilled: 작업이 성공적으로 완료. resolve()를 호출하여 완료한다. Promise를 사용하는 쪽에서는 then()으로 결과가 전달된다.
  - Reject: 실패. reject()를 호출한다. 사용자에게는 catch() 로 에러를 전달한다.

Promise

```
function test(value){
  return new Promise((resolve, reject)=>{
    if(value>=0)
      resolve('ok');
    else
      reject('error');
  });
}

test(1)
.then((result)=>{ console.log(result); })
.catch((error)=>{ console.log(error); })
```

```
function test(value){
  return new Promise((resolve, reject)=>{
    if(value>=0)
      resolve('ok');
    else
      reject('error');
  });
}

function test2(value){
  return new Promise((resolve, reject)=>{
    console.log('test2 ' + value);
    resolve('finished');
  });
}

function test3(value){
  console.log(value);
}

test(1)
.then(test2)
.then(test3)
.catch((error)=>{ console.log(error); })
```

Async, await

ECMAScript 2017 (ES8)

- Promise는 코드 스타일이 일반 함수와 다른 점이 단점.
- 일반 함수와 비슷한 스타일로 비동기 함수를 실행할 수 있다.

```
function test(value){
  return new Promise((resolve, reject)=>{
      resolve('ok');
  });
}

async function runTest(){
  const result = await test(1);
  console.log(result);
}

runTest();
```

Async, await
ECMAScript 2017 (ES8)

- 예외 처리는 try, catch를 사용한다.

```
function test(value){
  return new Promise((resolve, reject)=>{
    if(value >=0)
      resolve('ok');
    else
      reject('ng');
  });
}

async function runTest(){
  try{
    const result = await test(-1);
    console.log(result);
  }catch(error){
    console.log(error);
  }
}

runTest();
```

Class

Class

- 특수한 함수. 선언문과 표현식으로 정의할 수 있다.
- 클래스 선언은 Hoisting이 되지 않는다.

```
1  // 선언문
2  class Rectangle{
3      constructor(width, height){
4          this.width = width;
5          this.height = height;
6      }
7  }
8
9  // 표현식
10 const Circle = class {
11     constructor(radius) {
12         this.radius = radius;
13     }
14 }
15
16 const r = new Rectangle(10, 10);
17 const c = new Circle(5);
18 console.log(r);
19 console.log(c);
```

static

- 클래스의 정적 메소드/속성 정의.
- 객체(instance)에서는 호출 할 수 없다.

```
1  // 선언문
2  class Rectangle{
3      constructor(width, height){
4          this.width = width;
5          this.height = height;
6      }
7      // static property
8      static typeName='Rectangle';
9      // static method
10     static info(){
11         return `Type: ${Rectangle.typeName}`;
12     }
13     // instance method
14     getArea() {
15         return this.width * this.height;
16     }
17 }
18
19 const r = new Rectangle(10, 10);
20 console.log(r.getArea()); // call instance method
21 console.log(Rectangle.typeName);
22 console.log(Rectangle.info()); // call static(Class) method
```

상속

- extends 키워드로 상속. super 키워드로 부모의 속성을 사용할 수 있다.

```
1  class Position {
2      constructor(x, y){
3          this.x = x;
4          this.y = y;
5      }
6  }
7
8  class Position3D extends Position {
9      constructor(x, y, z){
10         super(x, y)
11         this.z = z
12     }
13 }
14
15 const p3 = new Position3D(0, 0, 0);
16 console.log(p3)
```

상속

- 함수 기반의 클래스에서도 상속 받을 수 있다.

```
function Position(x, y) {
    this.x = x;
    this.y = y;
}

class Position3D extends Position {
    constructor(x, y, z){
        super(x, y)
        this.z = z
    }
}

const p3 = new Position3D(0, 0, 0);
console.log(p3)
```