

JavaScript

module – 김근형 강사

Module

○ Module

JavaScript

Module

Cheatsheet



Module

○ Module

- 과거의 Javascript 는 그렇게 많은 프로그래밍이 필요하지 않았으며 지금처럼 프레임워크나 많은 라이브러리가 존재하지도 않았었다.
- node.js의 등장과 ECMAScript6 이후에 자바스크립트가 거대화 되면서 자바스크립트의 사용법이 확장됨과 동시에 여러 라이브러리, 프레임워크의 등장으로 자바스크립트 코드가 전체적으로 비대해지게 됨
- 효율적인 관리를 위해 별도의 단위로 쪼개 관리하는 기술이 필요해지게 되었으며 이런 기술을 모듈이라는 이름으로 지원하게 됨
- 모듈은 크게 CommonJS(Node.js) 방식의 모듈 작성 방식이 있고 ESM(ECMAScript Module) 방식의 모듈 작성 방식이 존재한다.

Module

○ CommonJS vs ESM(ECMAScript Module)

CommonJS

```
require("모듈명")  
module.exports
```

ESM

```
import  
export
```

Module

○ CommonJS Module

- CommonJS에서 사용되는 모듈 사용 방식
- Node.js에서 이 방법을 많이 사용한다.
- 단 일반 웹에서는 지원되지 않으므로 주의할 것

01-exports1.js

```
const PI = 3.14;
const getCirleArea = r => r * r * PI;

// 실제 내보내고자 하는 내용들을 module.exports 혹은
// exports 를 통해 내보낼 수 있다.

exports.PI = PI;
...
module.exports.getCirleArea = getCirleArea;
```

01-require1.js

```
// exports를 통해 넘긴 내용들은 다른 파일에서
// require를 통해 호출하여 사용이 가능하다.
// 보통 require 안에는 파일 명을 써주는데 뒤에
// .js는 생략한다.
const gca = require("./01-exports1");

console.log(gca.PI);
console.log(gca.getCirleArea(gca.PI));
```

Module

- CommonJS Module

- CommonJS 방식으로 모듈을 내보낼 때는 ES6처럼 명시적으로 선언하는 것이 아니라 특정 변수나 그 변수의 속성으로 내보낼 객체를 세팅해줘야 한다.
- 특히, 제일 헷갈리는 부분이 바로 유사해보이는 `exports` 변수와 `module.exports` 변수를 상황에 맞게 잘 사용해야 한다.
 - 여러 개의 객체를 내보낼 경우, `exports` 변수의 속성으로 할당한다.
 - 딱 하나의 객체를 내보낼 경우, `module.exports` 변수 자체에 할당한다.

Module

○ CommonJS Module - exports 예제

```
const exchangeRate = 0.91

function roundTwoDecimals(amount) {
  return Math.round(amount * 100) / 100
}

const canadianToUs = function (canadian) {
  return roundTwoDecimals(canadian * exchangeRate)
}

function usToCanadian(us) {
  return roundTwoDecimals(us / exchangeRate)
}

exports.canadianToUs = canadianToUs // 내보내기 1
exports.usToCanadian = usToCanadian // 내보내기 2
```

```
const currency = require("./02-exports2")

console.log("50 Canadian dollars equals this amount of US dollars:")
console.log(currency.canadianToUs(50))

console.log("30 US dollars equals this amount of Canadian dollars:")
console.log(currency.usToCanadian(30))
```

```
50 Canadian dollars equals this amount of US dollars:
45.5
30 US dollars equals this amount of Canadian dollars:
32.97
```

Module

○ CommonJS Module – module.exports 예제

```
const exchangeRate = 0.91

// 안 내보냄
function roundTwoDecimals(amount) {
  return Math.round(amount * 100) / 100
}

// 내보내기
const obj = {}
obj.canadianToUs = function (canadian) {
  return roundTwoDecimals(canadian * exchangeRate)
}
obj.usToCanadian = function (us) {
  return roundTwoDecimals(us / exchangeRate)
}
module.exports = obj
```

```
const currency = require("./03-exports3")

console.log("50 Canadian dollars equals this amount of US dollars:")
console.log(currency.canadianToUs(50))

console.log("30 US dollars equals this amount of Canadian dollars:")
console.log(currency.usToCanadian(30))
```

```
50 Canadian dollars equals this amount of US dollars:
45.5
30 US dollars equals this amount of Canadian dollars:
32.97
```


Module

- ESM Module

- ESM 모듈은 기본적으로 JavaScript 코드를 담고 있는 파일을 의미한다.
- 다만 일반적인 JavaScript 파일과는 다른 여러가지 차이점을 갖고 있다.
 - import 혹은 export 구문을 사용할 수 있다.
 - 별다른 처리를 해주지 않아도 엄격 모드(strict mode)로 동작한다.
 - 모듈의 가장 바깥쪽에서 선언된 이름은 전역 스코프가 아니라 모듈 스코프에서 선언된다.

Export & Import

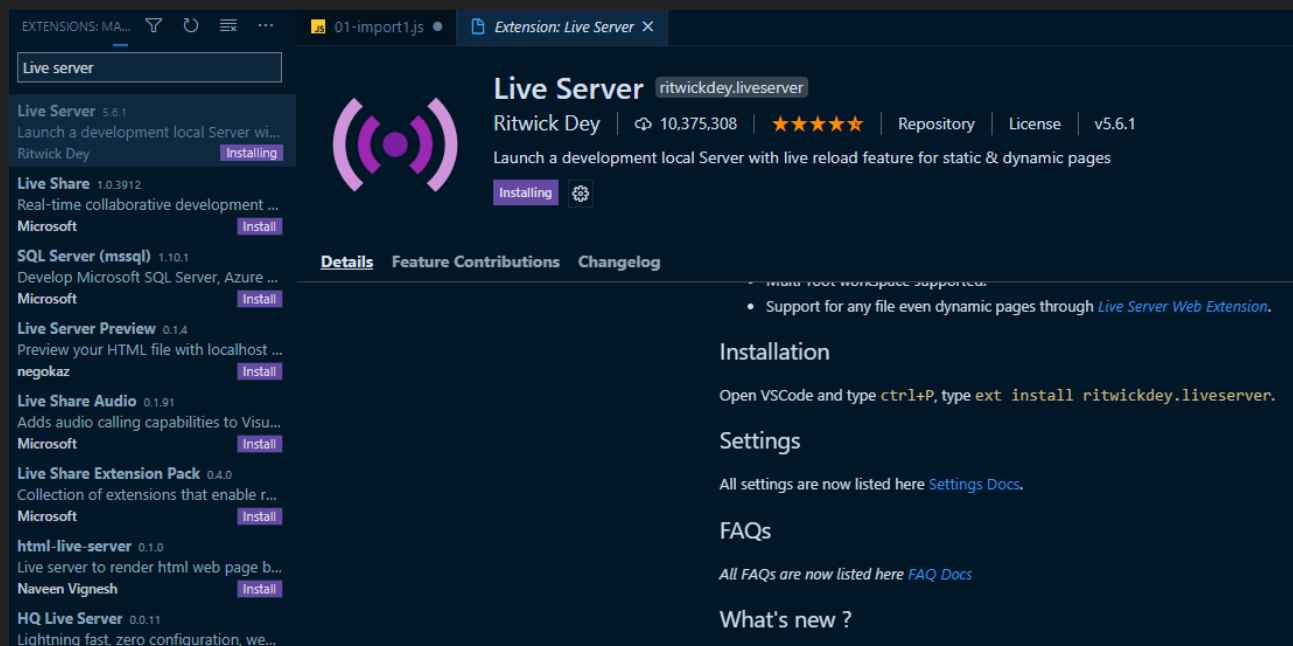
- export & import
 - 모듈에는 특수한 지시자 export와 import를 적용할 수 있다.
 - 위의 지시자를 사용하면 불러온 모듈에 있는 함수를 호출하는 것과 같은 기능 공유가 가능하다.

import	export
외부 모듈의 기능을 가져올 수 있다	외부 모듈에서 해당 지시자를 사용한 변수나 함수, 클래스에 접근 가능하다.

Export & Import

○ 테스트 전 팁

- 모듈은 CORS(Cross-Origin Resource Sharing) 제약으로 실행이 안될 가능성이 높기 때문에 반드시 Live Server Extension 과 같은 장치를 사용해서 실행한다.
- node.js와 같은 곳에서는 mjs 모듈을 설치하거나 아니면 package.json에 "type" : "module" 옵션을 넣어 주어야만 한다.



Export & Import

○ Export & Import 예제 - 1

01-export1.js

```
// export 하고자 하는 대상 앞에 export 지시자를 쓴다.  
export const a = 'hello';
```

01-import1.js

```
// export 하는 모듈을 불러올 경우  
// import {불러오고자 하는 변수, 함수, 클래스} from '파일명'  
// 형태로 불러올 수 있다.  
import {a} from './01-export1.js'  
  
console.log(a); // hello
```

01-ModuleEx1.html

```
<body>  
  <!-- 스크립트 타입에는 모듈을 불러오기 위해 반드시 module 이라 명시한다  
  |   | 명시하지 않을 경우 에러가 발생한다. -->  
  <script type="module" src="./01-import1.js"></script>  
</body>
```

hello

Export & Import

○ Export & Import 예제 - 2

02-ModuleEx2.html

```
<body>
  <script type="module">
    import { baz, func1, Rectangle } from './02-export2.js'

    console.log(baz);
    func1();
    let rec = new Rectangle(10, 10);
    console.log(rec.area)
  </script>
</body>
```

4321

이것은 func1의 기능입니다

100

02-export2.js

// 모듈은 변수, 함수, 클래스가 가능하다.

```
export const baz = 4321;
```

```
export function func1(){
  console.log('이것은 func1의 기능입니다');
}
```

```
export class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  // Getter
  get area() {
    return this.calcArea();
  }
  // 메서드
  calcArea() {
    return this.height * this.width;
  }
}
```

Export & Import

○ Export & Import 예제 - 3

03-export3.js

```
// 일일이 export를 선언하고 싶지 않다면  
// 변수나 함수, 클래스를 선언 후에 밑에서  
// export에 묶어 한꺼번에 선언할 수 있다.
```

```
let value1 = 'abcd';
```

```
function func1(){  
  console.log('func1 output');  
}
```

```
function func2(){  
  console.log('func2 output');  
}
```

```
export {value1, func1, func2}
```

03-ModuleEx3.html

```
<body>  
  <script type="module">  
    import {value1, func1 , func2} from './03-export3.js'  
  
    console.log(value1);  
    func1();  
    func2();  
  </script>  
</body>
```

abcd

func1 output

func2 output

Export & Import

○ Export & Import 예제 - 4

04-ModuleEx4.html

```
<script type="module">
  // import 문 작성시에도 as를 통해 기존 export에서 나온 이름을 바꿀 수가 있다
  // import시 바뀐 이름으로 사용이 가능하다.
  import {func1 as f1, func2 as f2, func3 as f3} from './04-export4.js'

  f1();
  f2();
  f3();
</script>
```

hardnameFunc1를 호출

hardnameFunc2를 호출

hardnameFunc3를 호출

04-export4.js

```
// export나 import시 해당 이름을 바꿀 수 있다.
// 이름을 바꿀때는 export {원래이름 as 바꿀이름} 으로
// 기존의 이름을 바꿀수가 있으며
// 이름을 바꾸는건 export, import에서 둘 다 가능하다.
```

```
function hardnameFunc1(){
  console.log('hardnameFunc1을 호출');
}
```

```
function hardnameFunc2(){
  console.log('hardnameFunc2를 호출');
}
```

```
function hardnameFunc3(){
  console.log('hardnameFunc3를 호출');
}
```

```
export {hardnameFunc1 as func1,
  hardnameFunc2 as func2,
  hardnameFunc3 as func3}
```

Export & Import

○ Export & Import 예제 - 5

05-ModuleEx5.html

```
<script type="module">
  // 만약 import 문 작성 시 해당 파일의 모든 기능들을 하나의 이름으로 담아
  // 쓰고 싶다면 '*' 를 이용해 선언이 가능하다
  // 단 합쳐진 형태는 하나의 객체 형태를 띄게 된다
  import * as ff from './05-export5.js'

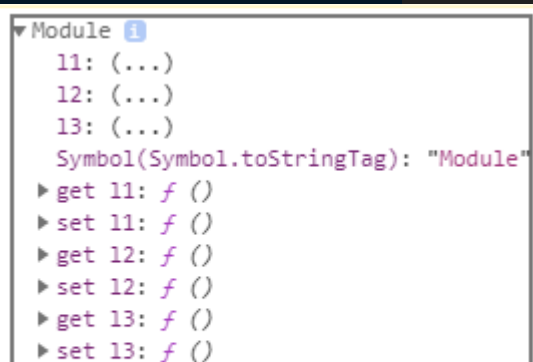
  console.dir(ff);

  console.log(ff.11);
  console.log(ff.12);
  console.log(ff.13);
</script>
```

05-export5.js

```
let 11 = 1;
let 12 = 2;
let 13 = 3;

export {11,12,13}
```



```
▼ Module 1
  11: (...)
  12: (...)
  13: (...)
  Symbol(Symbol.toStringTag): "Module"
  ▶ get 11: f ()
  ▶ set 11: f ()
  ▶ get 12: f ()
  ▶ set 12: f ()
  ▶ get 13: f ()
  ▶ set 13: f ()
```

1

2

3

Export & Import

○ default export

- export 에서 명명을 하지 않고 임의의 함수, 변수, 값, 클래스 등을 가져오고자 할 때 export 되는 개체를 default 로 선언하여 가져올 수 있다.
- default는 export 모듈 파일에 단 한 개만 존재할 수 있으며 복수 개 선언이 불가능하다.
- default 를 선언한 export 파일 내에 naming 된 함수 및 클래스 선언이 가능하다.
- default 선언한 개체를 가져오기 위해 import 문에서 {default as 이름} 형태로 가져올 수 있으며 아니면 name외 이름을 선언하여 가져올 수 있다.

Export & Import

○ default export 예제 - 1

06-DefaultModuleEx1.html

```
<script type="module">
  // 클라이언트에서는 default 개체를 불러올 때 {default as 선언할이름}
  // 을 붙여 사용이 가능하며 혹은 이름만 붙여도 사용이 가능하다.
  // import {default as foo} from './06-default1.js'
  import foo from './06-default1.js'

  console.log(foo);
</script>
```

bar

06-default1.js

```
// default 를 선언하기 위해서는 export 뒤에 default
// 라는 지시어를 붙여 사용이 가능하다.
export default 'bar';

// default 개체는 한 파일안에 두개가 존재할 수 없다.
// export default 'nam';
```

Export & Import

○ default export 예제 - 2

07-DefaultModuleEx2.html

```
<script type="module">
  // import에서 default 와 일반 기능을 같이 호출이 가능하며
  // 이럴 경우 반드시 default 개체는 as 명령어를 써서 이름을
  // 재 지정 해주어야만 한다
  import {default as f0, f1} from './07-default2.js'

  f0();
  f1();
</script>
```

이것은 default function 입니다.

이것은 명명한 function 입니다

07-default2.js

```
// default 선언이 가능한 것은 export로 선언될 수 있는
// 모든 개체가 된다. 단, export는 반드시 해당 파일에 하나만 선언될
// 수 있으므로 주의
export default function(){
  console.log('이것은 default function 입니다.');
```

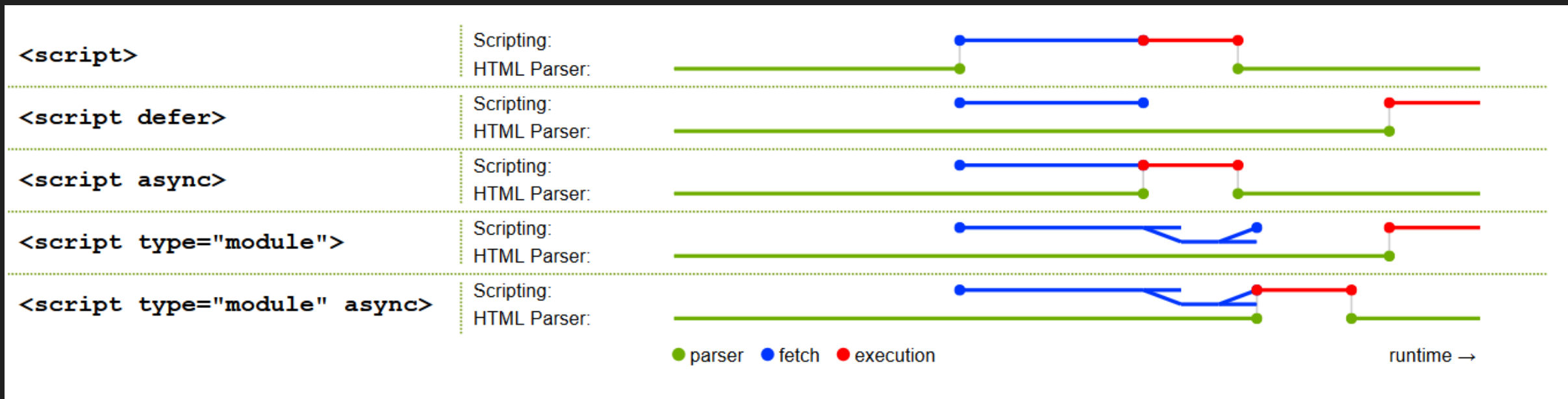
}

// default 를 선언한 곳에 name이 있는 개체를 선언해도 상관은 없다.
export function f1(){
 console.log('이것은 명명한 function 입니다');

}

Export & Import

○ 모듈 동기/비동기



Export & Import

- 모듈 동기/비동기

- `<script type="module">` 은 기본적으로 `<script defer>`와 동일한 성질을 가진다.
- 즉 import 할 스크립트를 전부 읽은 후에 html parsing이 완료되면 javascript 소스를 실행시킨다.
- 그렇기 때문에 `<script type="module">` 은 html head에 선언하여도 크게 관계는 없다.
- 단 모듈을 실시간으로 불러오고 싶다면 `async` 옵션을 추가할 수 있으며 `async` 옵션은 기존의 `<script async>`와 동일한 성질을 가진다.

Export & Import

○ 모듈 동기/비동기 예제

```
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script type="module">
    import {default as f1} from './08-asyncmodule.js'

    f1();
  </script>
  <title>Document</title>
</head>
<body>
  <p id="str"></p><br>
  <button id="btn" type="button">클릭</button>
</body>
```

```
export default function(){
  document.getElementById("btn").addEventListener("click", ()=>{
    document.getElementById("str").innerHTML = "hello"
  }, false);
}
```

hello

클릭