48. 모듈

모듈의 일반적인 의미

- 재사용 가능한 코드 조각
- 독립적인 파일 스코프(모듈 스코프) 존재
- 보통 기능을 기준으로 파일 분리
- 모듈의 자산은 캡슐화되어 다른 모듈에서 접근 불가
  - export를 통해 선택적으로 공개
  - 모듈 사용자(module consumer)는 import를 통해 사용

모듈의 이점

- 기능을 기준으로 파일 분리 → 코드의 단위를 명확히 분리
- 재사용성 ↑ → 개발 효율성 ↑ , 유지 보수성 ↑

자바스크립트와 모듈

자바스크립트는 웹페이지의 단순한 보조 기능을 처리하기 위한 제한적인 용도를 목적으로 태어남. 태생적 한계로 다른 프로그래밍 언어와 비교할 때 부족한 부분이 있는 것은 사실. 대표적인 것이 모듈 시스템 지원. 자바스크립트는 모듈이 성립하기 위해 필요한 파일 스코프와 import, export를 지원하지 않았음.

클라이언트 사이드 자바스크립트는 script 태그를 사용하여 외부의 자바스크립트 파일을 로드할 수 있지만, 파일마다 독립적인 파일 스코프는 존재하지 않았음. 이로 인해서 모든 자바스크립트 파일은 하나의 전역을 공유하는 문제 발생.

자바스크립트를 클라이언트 사이드, 즉 브라우저 환경에 국한하지 않고 사용하려는 움직임이 생기면서 모듈 시스템은 반드시 해결해야 하는 과제가 되었음. 이러한 상황에서 제안된 것이 CommonJS와 AMD다.

자바스크립트 모듈 시스템은 CommonJS와 AMD 진영으로 나뉘게 되었고, 브라우저 환경에서 모듈을 사용하기 위해서는 CommonJS 또는 AMD를 구현한 모듈 로더 라이브러리를 사용해야 했음.

Node.js는 모듈 시스템의 사실상 표준인 CommonJS를 채택하고, 독자적인 진화를 거쳐, CommonJS 사양을 따름. 즉, Node.js는 ECMAScript 표준 사양은 아니지만 모듈 시스템 지원. 따라서 Node.js 환경에서는 파일별로 독립적인 파일 스코프 가짐.

ES6 모듈

ES6에서는 클라이언트 사이드 자바스크립트에서도 동작하는 모듈 기능 추가

IE를 제외한 대부분의 브라우저(Chrome 61, FF 60, SF 10.1, Edge 16이상)에서는 ES6 모듈 사용 가능

사용법은 script 태그에 type=”module” 어트리뷰트를 추가하면, 자바스크립트 파일은 모듈로서 동작

일반적인 자바스크립트 파일이 아닌, ESM임을 명확히 하기 위해 파일 확장자는 mjs 권장

```
<script type="module" src="app.mjs"></script>
```

48.3.1 모듈 스코프

ESM이 아닌 일반적인 자바스크립트 파일은 script 태그로 분리해서 로드해도 독자적인 모듈 스코프를 갖지 않는다.

```
// foo.js
// x 변수는 전역 변수
var x = 'foo';
console.log(window.x); // foo
```

```
// bar.js
// x 변수는 전역 변수. foo.js에서 선언한 전역 변수 x와 중복된 선언이다.
var x = 'bar';
```

```
// foo.js에서 선언한 전역 변수 x 값이 재할당되었다.
console.log(window.x); //bar
```

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script src="foo.js"></script>
    <script src="bar.js"></script>
  </body>
</html>
```

ESM은 파일 자체의 독자적인 모듈 스코프 제공. 모듈 내에서 var 키워드로 선언한 변수는 전역 변수가 아니며, window 객체의 프로퍼티도 아니다

```
var x = "foo";
console.log(x); // foo
console.log(window); // undefined
```

```
var x = "bar";
console.log(x); // bar
console.log(window); // undefined
```

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script src="foo.js" src="foo.mjs"></script>
    <script src="bar.js" src="bar.mjs"></script>
  </body>
</html>
```

모듈 내에서 선언한 식별자는 모듈 외부에서 참조할 수 없다. 스코프가 다르기 때문이다

```
// foo.mjs
const x = 'foo';
console.log(x); // foo
```

```
// bar.mjs
console.log(x); // ReferenceError: x is not defined
```

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script src="foo.js" src="foo.mjs"></script>
    <script src="bar.js" src="bar.mjs"></script>
  </body>
</html>
```

48.3.2 export 키워드

모듈 내부에서 선언한 모든 식별자는 기본적으로 해당 모듈 내부에서만 참조할 수 있다. 모듈 내부에서 선언한 식별자를 외부에 공개하여 다른 모듈들이 재사용할 수 있게 하려면 export 키워드를 사용한다.

export 키워드는 선언문 앞에 사용한다.

```
// lib.mjs
// 변수의 공개
export const pi = Math.PI;
```

```
// 함수의 공개
export function = square(x) {
  return x * x;
}

// 클래스의 공개
export class Person {
  constructor(name) {
    this.name = name;
  }
}
```

선언문 앞에 매번 export 키워드를 붙이는 것이 번거롭다면, export할 대상을 하나의 객체로 구성하여 한번에 export 할 수 있다

```
// lib.mjs
const pi = Math.PI;

function = square(x) {
  return x * x;
}

class Person {
  constructor(name) {
    this.name = name;
  }
}

export { pi, square, Person };
```

48.3.3 import 키워드

다른 모듈에서 공개한 식별자를 자신의 스코프 내부로 로드하려면 import 키워드를 사용.

다른 모듈이 export한 식별자 이름으로 import해야 하며, ESM의 경우, 파일 확장자를 생략할 수 없다.

```
// app.mjs
// 같은 폴더 내의 lib.mjs 모듈이 export한 식별자 이름으로 import한다.
// ESM의 경우, 파일 확장자를 생략할 수 없다.

import { pi, square, Person } from './lib.mjs';

console.log(pi); // 3.1415
console.log(square(10); // 100
console.log(new Person('Lee')); // Person { name : 'Lee' }
```

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script type="module" src="app.mjs"></script>
  </body>
</html>
```

위 예제에서 app.mjs는 진입점(EntryPoint)이므로 반드시 script 태그로 로드해야 함.

하지만 lib.mjs는 app.mjs의 import 문에 의해 로드되는 의존성(dependancy)이다.

따라서 lib.mjs는 script 태그로 로드하지 않아도 된다.

모듈이 export한 식별자 이름을 일일이 지정하지 않고, 하나의 이름으로 한 번에 import할 수 도 있다. 이때 import 되는 식별자는 as 뒤에 짖어 한 이름의 객체에 프로퍼티로 할당된다.

```
// app.mjs
// lib.mjs 모듈이 export한 모든 식별자를 lib 객체의 프로퍼티로 모아 import 한다.

import * as lib from './lib.mjs';

console.log(lib.pi);
console.log(lib.square(10));
console.log(new lib.Person('lee'));
```

모듈이 export한 식별자 이름을 변경하여 import할 수도 있다.

```
// app.mjs
// lib.mjs 모듈이 export한 모든 식별자를 lib 객체의 프로퍼티로 모아 import 한다.

import { pi as PI, square as sq, Person as P} from './lib.mjs';

console.log(PI);
console.log(sq(10);
console.log(new P('lee'));
```

모듈에서 하나의 값만 export한다면, default 키워드를 사용할 수 있다.

```
// lib.mjs
export default x => x * x;
```

default 키워드를 사용하는 경우, var, let, const 키워드는 사용할 수 없다.

```
// lib.mjs
export default const foo = () => {};
// => SyntaxError: Unexpected token 'const'
```

default 키워드와 함께 export한 모듈은 {} 없이 임의의 이름으로 import한다

```
// app.mjs
import square from './lib.mjs';
console.log(square(3));
```