#웹팩이란?

웹팩이란 최신 프런트엔드 프레임워크에서 가장 많이 사용되는 모듈 번들러(Module Bundler)입니다. 모듈 번들러란 웹 애플리케이션을 구성하는 자원(HTML, CSS, Javscript, Images 등)을 모두 각각의 모듈로 보고 이를 조합해서 병합된 하나의 결과물을 만드는 도구를 의미합니다. 그럼 모듈과 모듈 번들링에 대해서 조금 더 살펴보겠습니다.

#모듈이란?

모듈이란 프로그래밍 관점에서 특정 기능을 갖는 작은 코드 단위를 의미합니다. 자바스크립트로 치면 아래와 같은 코드가 모듈입니다.

```
// math.js
function sum(a, b) {
  return a + b;
}

function substract(a, b) {
  return a - b;
}

const pi = 3.14;

export { sum, substract, pi }
```

이 `math.js` 파일은 아래와 같이 3가지 기능을 갖고 있는 모듈입니다.

1. 두 숫자의 합을 구하는 `sum()` 함수
2. 두 숫자의 차를 구하는 `substract()` 함수
3. 원주율 값을 갖는 `pi` 상수

이처럼 성격이 비슷한 기능들을 하나의 의미 있는 파일로 관리하면 모듈이 됩니다.

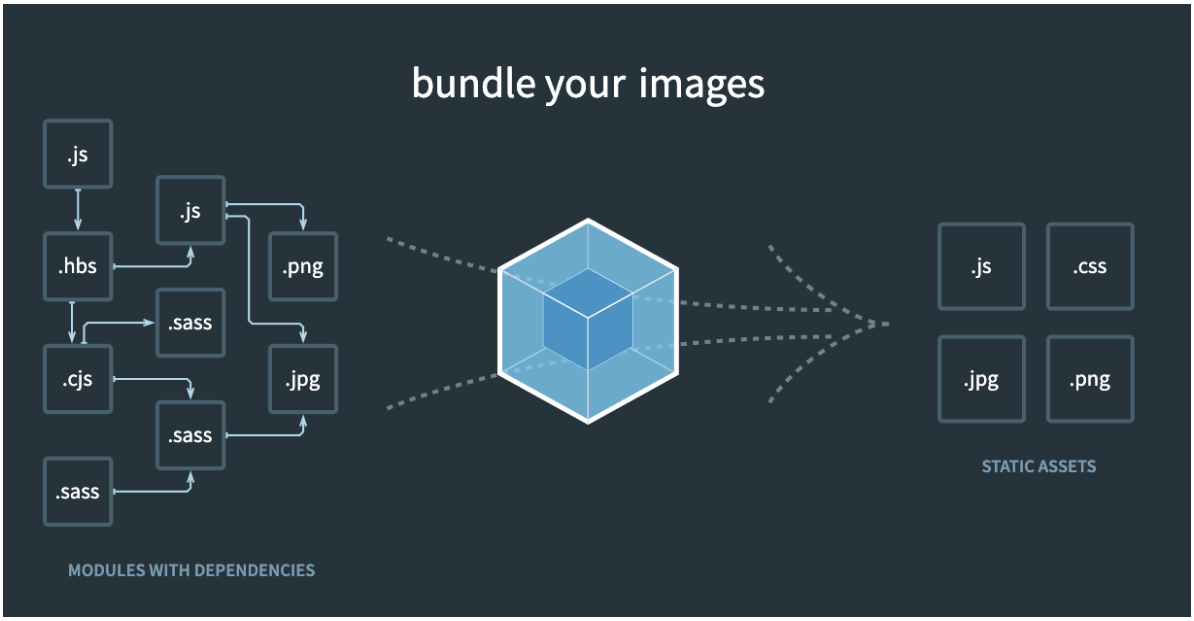
위의 `export` 코드는 ES6의 Modules 문법을 사용하였습니다. 문법을 잘 모르시는 분들은 여기를 참고하세요.

#웹팩에서의 모듈

웹팩에서 지칭하는 모듈이라는 개념은 위와 같이 자바스크립트 모듈에만 국한되지 않고 웹 애플리케이션을 구성하는 모든 자원을 의미합니다. 웹 애플리케이션을 제작하려면 HTML, CSS, Javascript, Images, Font 등 많은 파일들이 필요하죠. 이 파일 하나하나가 모두 모듈입니다.

#모듈 번들링이란?

아래 그림과 같이 웹 애플리케이션을 구성하는 몇십, 몇백개의 자원들을 하나의 파일로 병합 및 압축 해주는 동작을 모듈 번들링이라고 합니다.



빌드, 번들링, 변환 이 세 단어 모두 같은 의미입니다.

웹팩의 등장 배경

웹팩이 등장한 이유는 크게 3가지입니다.

1. 파일 단위의 자바스크립트 모듈 관리의 필요성
2. 웹 개발 작업 자동화 도구 (Web Task Manager)
3. 웹 애플리케이션의 빠른 로딩 속도와 높은 성능

#파일 단위의 자바스크립트 모듈 관리

입문자 관점에서 고안된 자바스크립트는 아래와 같이 편리한 유효 범위를 갖고 있습니다.

```
var a = 10;
console.log(a); // 10

function logText() {
  console.log(a); // 10
}
```

자바스크립트의 변수 유효 범위는 기본적으로 전역 범위를 갖습니다. 최대한 넓은 변수 범위를 갖기 때문에 어디에서도 접근하기가 편리하죠.

그런데 이러한 장점이 실제로 웹 애플리케이션을 개발할 때는 아래와 같은 문제점으로 변합니다.

```
<!-- index.html -->
<html>
  <head>
    <!-- ... -->
  </head>
  <body>
    <!-- ... -->
    <script src="./app.js"></script>
    <script src="./main.js"></script>
  </body>
```

```
</html>
// app.js
var num = 10;
function getNum() {
  console.log(num);
}
// main.js
var num = 20;
function getNum() {
  console.log(num);
}
```

위와 같이 `index.html`에서 두 자바스크립트 파일을 로딩하여 사용한다고 해봅시다. 스크립트에 아래와 같이 코드를 실행하면 어떤 결과가 나올까요?

```
<!-- index.html -->
<html>
  <head>
    <!-- ... -->
  </head>
  <body>
    <!-- ... -->
    <script src="./app.js"></script>
    <script src="./main.js"></script>
    <script>
      getNum(); // 20
    </script>
  </body>
</html>
```

결과는 20입니다. `app.js`에서 선언한 `num` 변수는 `main.js`에서 다시 선언하고 20을 할당했기 때문이죠.

이러한 문제점은 실제로 복잡한 애플리케이션을 개발할 때 발생합니다. 변수의 이름을 모두 기억하지 않은 이상 변수를 중복 선언하거나 의도치 않은 값을 할당할 수 있죠.

이처럼 파일 단위로 변수를 관리하고 싶은 욕구, 자바스크립트 모듈화에 대한 욕구를 예전까진 AMD, Common.js와 같은 라이브러리로 풀어왔습니다.

#웹 개발 작업 자동화 도구

이전부터 프런트엔드 개발 업무를 할 때 가장 많이 반복하는 작업은 텍스트 편집기에서 코드를 수정하고 저장한 뒤 브라우저에서 새로 고침을 누르는 것이었습니다. 그래야 화면에 변경된 내용을 볼 수 있었죠.

이외에도 웹 서비스를 개발하고 웹 서버에 배포할 때 아래와 같은 작업들을 해야 했습니다.

- HTML, CSS, JS 압축
- 이미지 압축
- CSS 전처리기 변환

이러한 일들을 자동화 해주는 도구들이 필요했습니다. 그래서 Grunt와 Gulp 같은 도구들이 등장했습니다.

#웹 애플리케이션의 빠른 로딩 속도와 높은 성능

일반적으로 특정 웹 사이트를 접근할 때 5초 이내로 웹 사이트가 표시되지 않으면 대부분의 사용자들은 해당 사이트를 벗어나거나 집중력을 잃게 됩니다.

그래서 웹 사이트의 로딩 속도를 높이기 위해 많은 노력들이 있었습니다. 그 중 대표적인 노력이 브라우저에서 서버로 요청하는 파일 숫자를 줄이는 것입니다. 이를 위해 앞에서 살펴본 웹 태스크 매니저를 이용해 파일들을 압축하고 병합하는 작업들을 진행했습니다.

뿐만 아니라 초기 페이지 로딩 속도를 높이기 위해 나중에 필요한 자원들은 나중에 요청하는 레이지 로딩(Lazy Loading)이 등장했죠.

웹팩은 기본적으로 필요한 자원은 미리 로딩하는게 아니라 그 때 그 때 요청하자는 철학을 갖고 있습니다.

웹팩으로 해결하려는 문제?

웹팩의 등장 배경에서도 살펴봤지만 웹팩에서 해결하고자 하는 기존의 문제점은 다음 4가지 입니다.

- 자바스크립트 변수 유효 범위
- 브라우저별 HTTP 요청 숫자의 제약
- 사용하지 않는 코드의 관리
- Dynamic Loading & Lazy Loading 미지원

#자바스크립트 변수 유효 범위 문제

웹팩은 변수 유효 범위의 문제점을 ES6의 Modules 문법과 웹팩의 모듈 번들링으로 해결합니다.

#브라우저별 HTTP 요청 숫자의 제약

TCP 스펙에 따라 브라우저에서 한 번에 서버로 보낼 수 있는 HTTP 요청 숫자는 제약되어 있습니다. 아래의 표는 최신 브라우저 별 최대 HTTP 요청 횟수입니다.

브라우저	최대 연결 횟수
익스플로러 7	2
익스플로러 8 ~ 9	6
익스플로러 10, 11	8, 13
크롬	6
사파리	6
파이어폭스	6
오페라	6
안드로이드, iOS	6

따라서, HTTP 요청 숫자를 줄이는 것이 웹 애플리케이션의 성능을 높여줄 뿐만 아니라 사용자가 사이트를 조작하는 시간을 앞당겨 줄 수 있죠.

⚠ 알아두면 좋아요!
클라이언트에서 서버에 HTTP 요청을 보내기 위해서는 먼저 TCP/IP가 연결되어야 합니다.

웹팩을 이용해 여러 개의 파일을 하나로 합치면 위와 같은 브라우저별 HTTP 요청 숫자 제약을 피할 수 있습니다.

#Dynamic Loading & Lazy Loading 미지원

Require.js와 같은 라이브러리를 쓰지 않으면 동적으로 원하는 순간에 모듈을 로딩하는 것이 불가능 했습니다. 그러나 이젠 웹팩의 Code Splitting 기능을 이용하여 원하는 모듈을 원하는 타이밍에 로딩할 수 있습니다.