

Front

도구/개발환경 1

유성민

2020

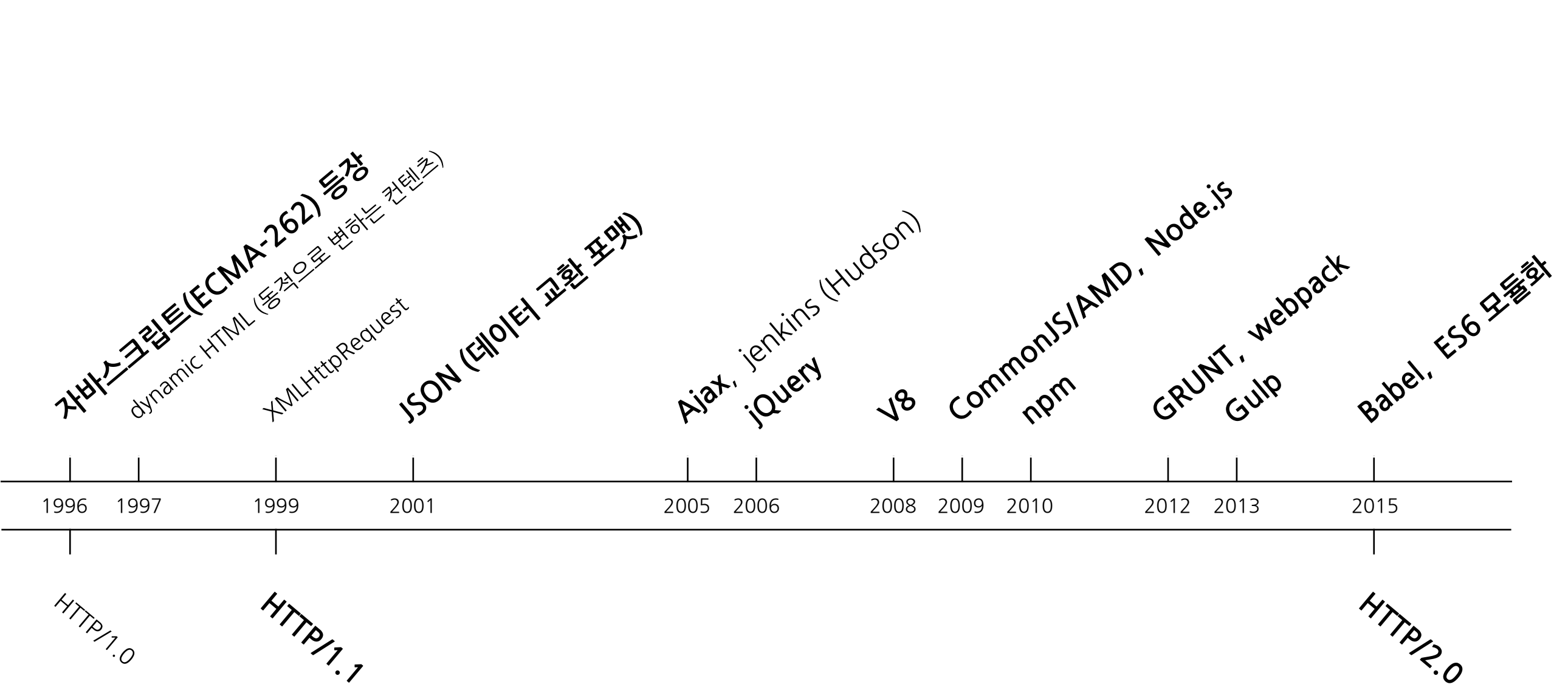


하림

우리는 Front 개발에 있어, 어떠한 이유로 그 도구를 사용(선택)하는가 ?

역사적 주요 흐름

Front 도구/개발환경



역사적 주요 흐름

Front 도구/개발환경

1996년 자바스크립트(ECMAScript 1.0) 등장

웹브라우저 안에서 돌아가며, 주로 웹페이지에 효과를 주거나 유효성검사 또는 단순 이벤트 처리
반짝이는 효과, 마우스 따라다니는 그림 등 개발을 한다는 시각보다는 주로 “copy & paste” 방식으로 사용

1997년 dynamic HTML

동적으로 변하는 콘텐츠
페이지의 문서 객체 모델(DOM)에 접근하여 콘텐츠와 스타일을 변화하고,
요소를 보여주거나 숨기는 것과 같은 것들을 가능
동적인 HTML은 IE4와 넷스케이프 네비게이터4에서 처음 등장

1999년 XMLHttpRequest

Internet Explorer 5에 비동기 웹 구현을 위한 XMLHttpRequest ActiveX를 탑재
Mozilla가 이 기술을 받아들여 2002년 출시한 Gecko Layout Engine 1.0에 적용하면서 XMLHttpRequest가 탄생
클라이언트 측에서 http 또는 https로 서버에게 요청(또는 전송)하고, 텍스트 형식으로 데이터를 받는 것을 가능
당시 웹이 가진 영향력은 현재에 비해 초라한 수준으로,
비동기 웹 기술이 가지고 있는 장점에도 불구하고, 큰 관심을 받지 못함

역사적 주요 흐름

Front 도구/개발환경

2001년 JSON, 자바스크립트 기반의 데이터 교환 포맷

더글라스 크락포드는 텍스트 형식으로 데이터를 저장하는 자바스크립트 문법을 'JSON'이라 이름 붙이고 문서화
JSON은 객체, 배열, 문자열, 숫자들, 불린 등을 구조적인 데이터로 표현하기 위해, 자바스크립트 리터럴로 사용

2005년 Ajax, 브라우저 기반의 어플리케이션 (XMLHttpRequest + dynamic HTML)

Ajax라는 용어는 Jesse James Garret가 Google Page에 사용한 기술을 소개한

“Ajax: A New Approach to Web Applications”라는 글에 처음 등장

2005년 2월, Ajax를 통해 달성할 수 있는 인상적인 예로 '구글 Maps' 소개

컨텐츠 로딩을 백그라운드에서 비동기적으로 진행 (XMLHttpRequest)

그 결과물을 가지고 현재 페이지에서 동적으로 업데이트 (dynamic HTML)

매번 페이지를 다시 Load 하는 것보다 많은 사용성 개선효과 (UX측면)

2006년 jQuery, Dom조작을 도와준다.

브라우저 DOM을 조작하는 것은 웹 개발에서 가장 힘든 부분 (브라우저 마다 제각각 대응)

jQuery는 DOM 조작을 위한 강력하고 매끄러운 API를 제공

HTTP/1.x

1996 ~ 2015

Front 도구/개발환경

HTTP/1.0 Connection 당 하나의 요청을 처리 하도록 설계

현재 요청에 대한 응답을 받은 후 다음 요청을 보내는, 기본적으로 HTTP 요청은 순차적

TCP연결 -> 요청 -> 응답 -> TCP종료 -> TCP연결 -> 요청 -> 응답 -> TCP종료 -> 요청 -> ...

HTTP 문서안에 포함된 다수의 리소스(Image, CSS, Script File) 개수에 비례해서 대기 시간(Latency) 길어짐
(네트워크 지연과 대역폭 제한에 걸려, 다음 요청을 보내는 데까지 상당한 딜레이가 발생할 수 있음)

HTTP/1.1 두 가지 모델이 추가됨

지속연결 (Persistent connection)

어느한쪽이 명시적으로 연결을 종료하지 않는 이상 **TCP 연결을 계속 유지**
(여러 요청과 응답에 대한 TCP 커넥션 연결/종료를 반복하는 부분을 줄여줌)

파이프라이닝 (HTTP Pipelining)

순차적인 HTTP 요청은 네트워크 지연과 대역폭 제한에 걸려,

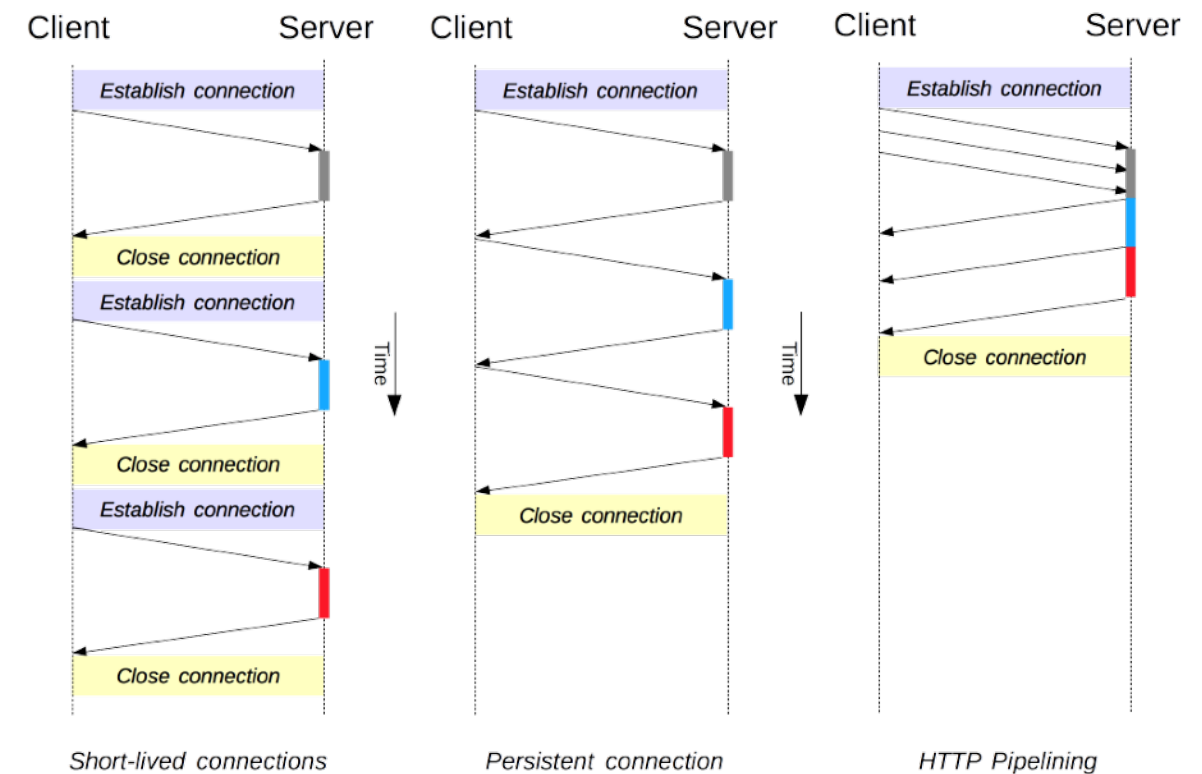
다음 요청을 보내는 데까지 상당한 딜레이가 발생할 수 있음

파이프라이닝은 **응답을 기다리지 않고 요청을 연속적으로 보내는 기능**

(하나의 응답 지연이 발생하면, 나머지 응답도 모두 지연)

(GET, HEAD, PUT, DELETE 메서드만 가능)

(브라우저에서 기본적으로 활성화되어있지 않음)



HTTP/1.x

Front 도구/개발환경

Domain Sharding - 제외된(deprecated)기술, HTTP/2 표준전환 권장함

HTTP/1.1 하나의 호스트당 2개의 동시 다운로드 허용

브라우저들은 각 도메인에 대한 몇 개의 커넥션을 맺고 병렬로 요청

Domain Sharding 숫자는 브라우저별로 각각 다르며,

대부분 Connections per Hostname 6개, Max Connections 17개

(hostname sharding 은 추가적인 DNS 조회를 발생시킴)

SPDY - Multiplexing

하나가 지연되면 나머지 응답도 모두 지연되는

FIFO(First In, First Out)처리 방식의

HTTP 파이프라이닝(pipelining)과는 달리,

각각의 요청, 응답이 모두 독립적으로 처리

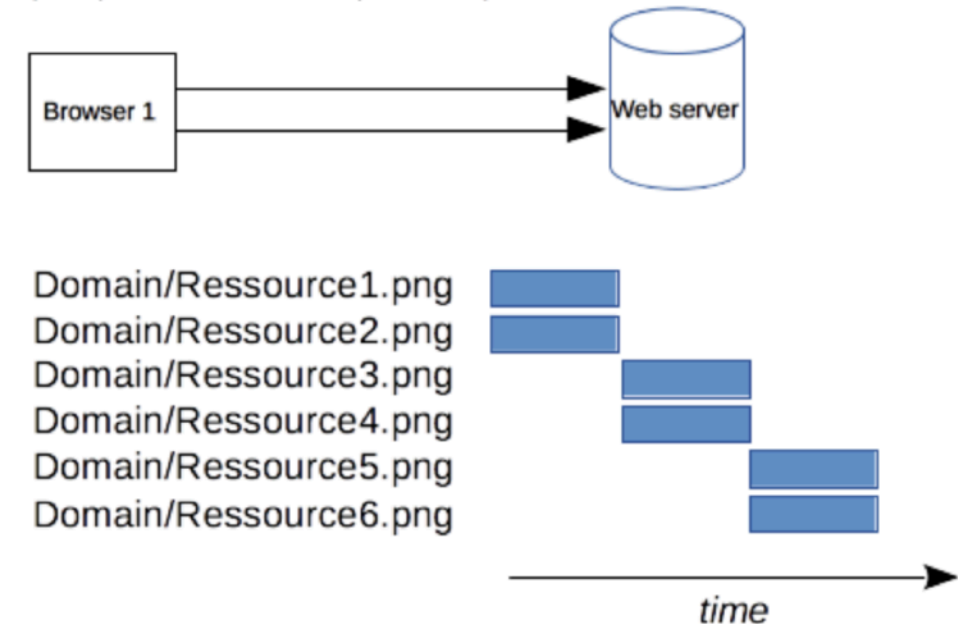
(‘speedy’라는 단어를 기반으로 Google이 만든 조어)

(SPDY를 쓰는 경우에는 hostname sharding이 불필요)

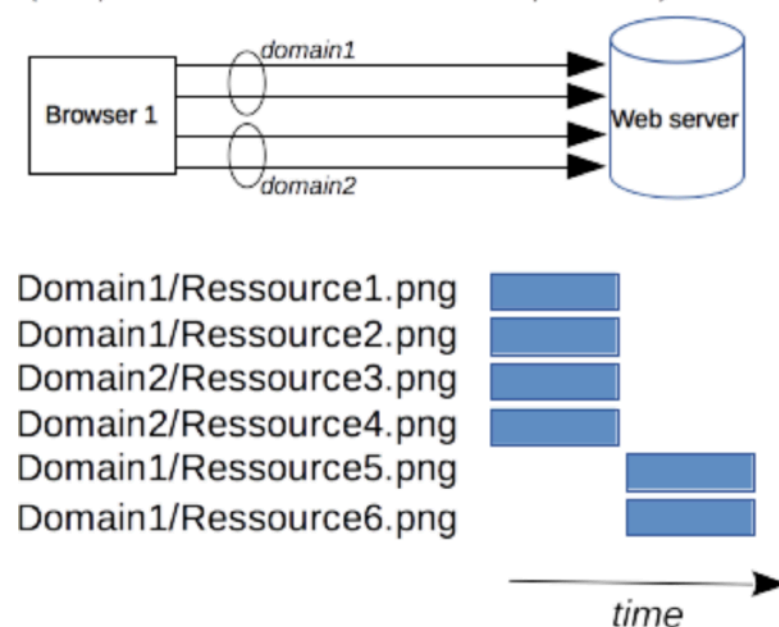
(SPDY는 Google에서 개발한 비표준 프로토콜, HTTP/2.0 표준화 기반이 됨)

1996 ~ 2015

Without domain sharding
(example with 2 connections per domain)



With domain sharding
(example with 2 domains and 2 connections per domain)



HTTP/1.x

1996 ~ 2015

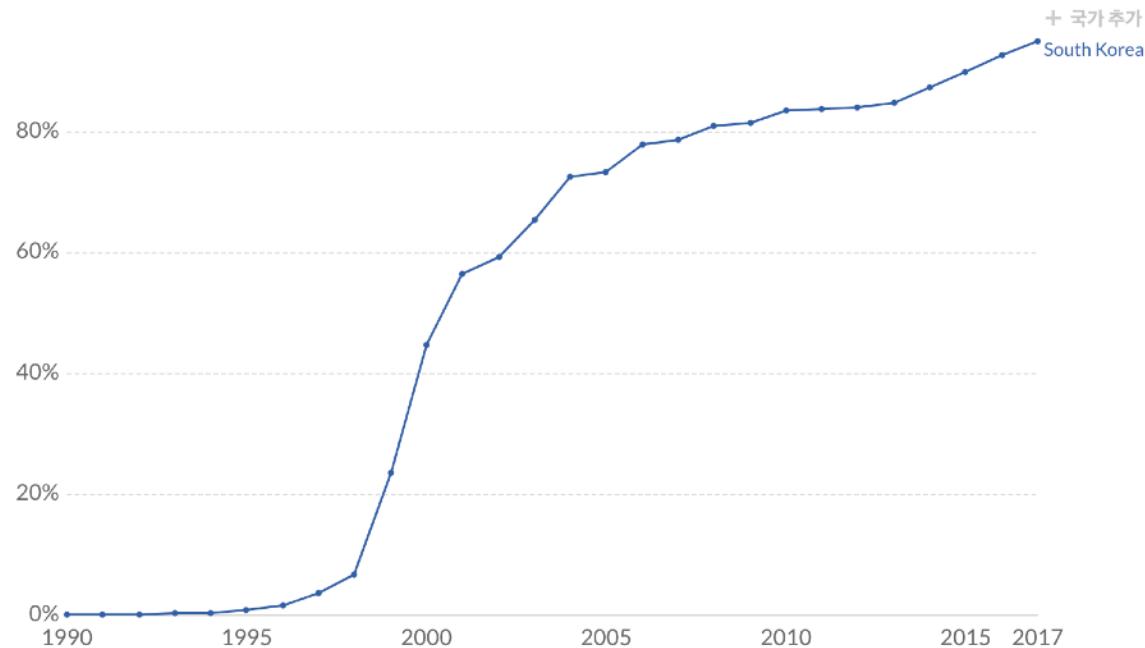
Front 도구/개발환경

Google I/O 2012에서 Google이 발표한 자료에 의하면,
1996년도의 Yahoo 메인 페이지 용량은 34KB 정도, 2012년 기준 평균 웹 페이지 용량의 30분 1 수준
1990년대의 웹 페이지에 비하여, 그 규모와 페이지당 HTTP 요청 개수가 20배 가량 증가

인터넷을 사용하는 인구의 비율

지난 3개월 동안 인터넷을 사용한 모든 개인은 인터넷 사용자로 계산됩니다. 인터넷은 컴퓨터, 휴대 전화, 개인 휴대 정보 단말기, 게임기, 디지털 TV 등을 통해 사용할 수 있습니다.

Our World
in Data



HTTP는 1991년에 0.9 버전이 처음 발표(비표준)된 후,
1996년에 1.0 버전, 1999년에 1.1 버전이 표준화되었으며,
이후 10년이 넘는 시간 동안 전혀 변화가 없었음

초고속 인터넷 보급과 함께 점점 더 미려한 사용자 UX 덕분에
페이지 크기와 요청 개수의 평균 값이 빠르게 증가했으나,
HTTP 네트워크 표준은 발전없이 머물러 있었음

[인터넷 사용 인구 비율] <https://ourworldindata.org/internet>

[HTTP 요청 통계] <https://httparchive.org/>

Minify

Front 도구/개발환경

JavaScript / CSS 코드 압축(minify) 도구

자바스크립트 코드에 들어있는 주석, 공백 등을 제거

2001년 JSMin

더글라스 크록포드(Douglas Crockford)가 만든 JavaScript Minifier
오픈소스로 C, C#, Java, JavaScript, Perl, Python, Ruby로 제공

2004년 packer

웹상에서 간단하게 사용할 수 있다는게 큰 장점

- 참고 내용

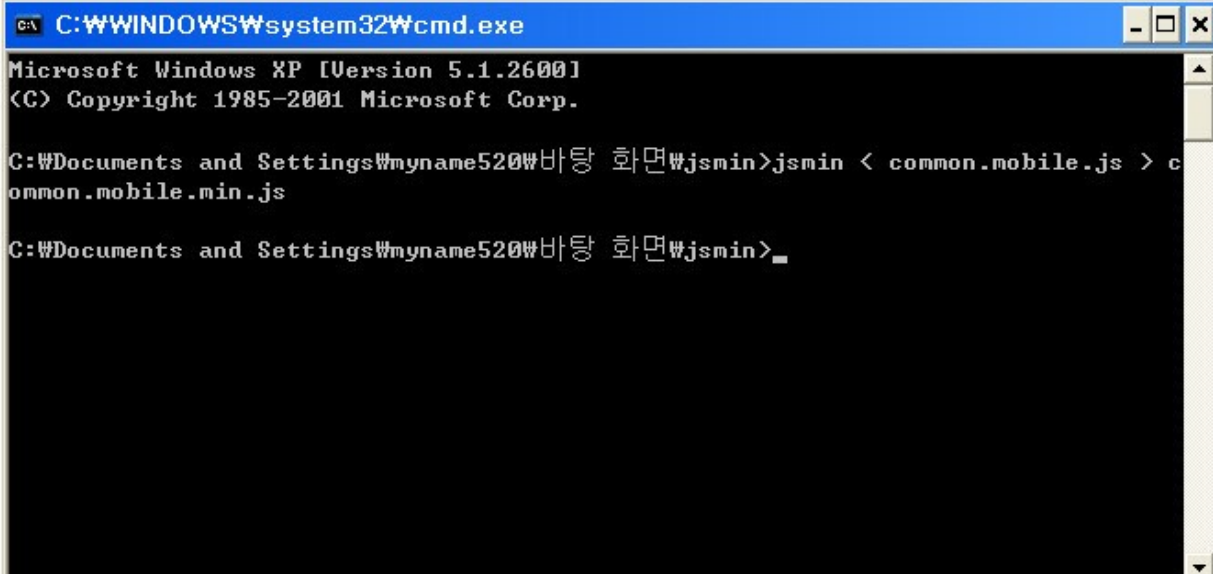
YUI Compressor (JavaScript 뿐만 아니라 CSS도 지원)

Yahoo User Interface Library에서 제공

jar파일 형태로 제공되고 있고 사용하려면 자바 1.4이상이 필요

Online GUI 형태의 YUI Compressor Online 있음

네트워크 대응을 위한 노력으로 이미지 압축도 있음(Image Spriting)



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\myname520\바탕 화면\jsmin>jsmin < common.mobile.js > c
ommon.mobile.min.js

C:\Documents and Settings\myname520\바탕 화면\jsmin>
```

\$ jsmin <압축할파일> 새로만들 파일명 "주석"

[JSMin] <https://www.crockford.com/jsmin.html>

[packer] <http://dean.edwards.name/download/#packer>

역사적 주요 흐름

Front 도구/개발환경

2008년 V8, 더 빨라진 자바스크립트

구글이 크롬 웹 브라우저를 소개했을 때, 많은 장점들 중 하나는 V8 이라고 불리는 빠른 자바스크립트 엔진
이는 자바스크립트가 느리다는 인식을 바꾸고, 다른 브라우저 제조사들과의 속도 경쟁을 이끌게 됨
V8 엔진의 등장은 서버사이드 JavaScript 진영(JavaScript를 브라우저 밖에서도 사용하려는 노력)에도 활기를 불어넣음

2009년 CommonJS/AMD, 자바스크립트 모듈화

JavaScript를 브라우저에서만 아니라,
서버사이드 애플리케이션이나 데스크톱 애플리케이션에서도 사용하려고 조직한 자발적 워킹 그룹 (브라우저 밖에서도 JavaScript 사용하자는 목표)
CommonJS의 모듈 명세는 모든 파일이 로컬 디스크에 있어 필요할 때 바로 불러올 수 있는 상황을 전제 (모듈 사전 설치 후 사용)

CommonJS의 주요 명세

- 스코프(Scope): 모든 모듈은 자신만의 독립적인 실행 영역이 있어야 한다.
- 정의(Definition): 모듈 정의는 exports 객체를 이용한다.
- 사용(Usage): 모듈 사용은 require 함수를 이용한다.

AMD 그룹은 비동기 상황에서도 JavaScript 모듈을 쓰기 위해 CommonJS에서 함께 논의하다 합의점을 이루지 못하고 독립한 그룹
AMD가 목표로 하는 것은 필요한 모듈을 네트워크를 이용해 내려받아야 하는 브라우저 환경(비동기)에서도 모듈을 사용할 수 있도록 표준을 만드는 일

CommonJS/AMD 모두 서버환경, 브라우저환경에서 사용가능한 모듈화 방식을 각각 추가 정의되어 있음
(JavaScript를 브라우저 밖으로 꺼내 사용하기 위한 노력의 일환 ConnomJS, 브라우저 내에서 모듈 실행에 중점을 두었던 AMD, 탄생배경 차이)

CommonJS / AMD Module

Front 도구/개발환경

CommonJS

ysm.js

```
// $ npm install jquery
// node_module/jquery
var $ = require('jquery');
var ysm = '유성민';
module.exports = {
  a: $,
  b: ysm
};
```

index.js

```
var ysmReturnValue = require('./ysm');
console.log(ysmReturnValue.a); // jquery
console.log(ysmReturnValue.b); // 유성민
```

AMD 라이브러리

```
<script src="require.js"></script>
```

ysm.js

```
define(['jquery'], function($) {
  console.log($);
  var ysm = '유성민';
  return {
    a: $,
    b: ysm,
  };
});
```

index.js (ysm 모듈 동적 로딩)

```
require(['ysm'], function(ysmReturnValue) {
  console.log(ysmReturnValue.a); // jquery
  console.log(ysmReturnValue.b); // 유성민
});
```

CommonJS 와 AMD 둘 모두 브라우저에서 종속된 모듈들을 불러올 때, (비동기 방식형태)
필요로 하는 JS 파일(모듈)들이 많아 질 수록 성능이슈(네트워크) 및 페이지 작업(동적화면의 경우)이 느려질 수 있음
즉, 번들링(Bundling) 작업이 필요한 이유

역사적 주요 흐름

Front 도구/개발환경

2009년 Node.js, 서버 사이드의 자바스크립트

이벤트 방식의 non-blocking I/O

자바스크립트(V8), CommonJS 명세의 모듈화 사용

클라이언트와 서버를 같은 언어로 구현가능

```
// 5 line 으로 쉽게 서버구축
var http = require('http');
http.createServer(function(request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World');
}).listen(80);
```

2010년 NPM, 노드 패키지 관리

Node.js 에서 사용할 수 있는 모듈들을 패키지화하여 모아둔 저장소 역할과

패키지 설치 및 관리를 위한 CLI (Command line interface)를 제공

package.json

패키지들은 버전이 빈번하게 업데이트, 각각 프로젝트가 의존하고 있는 패키지들에 대해 관리 필요

npm 은 package.json 파일을 통해서 프로젝트의 정보와 패키지 의존성(dependency)을 관리 (프로젝트가 사용하는 모듈 리스트/버전)

이미 의존성을 명시한 package.json 이 존재 한다면, 팀 내 배포를 통해 동일한 개발 환경을 빠르게 구축할 수 있다는 장점

(해당 프로젝트가 의존하는 패키지들의 이름과 버전을 명시하고, npm install 명령을 통해 의존 모듈 리스트 한번에 설치)

(package.json 은 Java 의 maven pom.xml 을 통해 프로젝트 의존성을 명시한다는 부분과 닮아 있음)

```
$ npm init
```

→ 질문/답 형태로 프로젝트 정보를 입력받아 package.json 생성가능

```
$ npm install <package>
```

→ /node_modules/ 폴더에 설치됨 (프로젝트 또는 전역설치 옵션존재)
package.json 파일이 있을 경우 '\$ npm install' 명령으로 명시된 종속 항목을 설치

Task Runner

Front 도구/개발환경

2012년 GRUNT, 태스크 러너 (2013년 Gulp)

사전 정의된 작업(task) 단위를

command line 명령을 통해 자동 실행시키는

javascript 빌드 도구 (자동화도구)

예를 들어 concatenating, minifying, validating 등

반복되는 작업을 task 로 정의하고,

명령어 하나로 실행함으로써 작업효율을 높여주는 도구

```
$ grunt <task>
```

Gruntfile.js

```
module.exports = function(grunt) {  
  // grunt 설정  
  grunt.initConfig({  
    pkg: grunt.file.readJSON('package.json'),  
    // uglify 설정  
    uglify: {  
      build: {  
        // uglify 타겟 설정  
        src: 'dist/build/result.js',  
        // uglify 결과 파일  
        dest: 'dist/build/result.min.js'  
      },  
    },  
    // concat 설정  
    concat: {  
      basic: {  
        // concat 타겟 설정 (앞에서부터 순서대로 합쳐진다.)  
        src: ['src/js/util.js', 'src/js/app.js'],  
        // concat 결과 파일  
        dest: 'dist/build/result.js'  
      },  
    },  
  });  
  // plugin 로딩  
  grunt.loadNpmTasks('grunt-contrib-uglify');  
  grunt.loadNpmTasks('grunt-contrib-concat');  
  // $ grunt concat - 명령어로 task 선택 실행  
  grunt.registerTask('concat', ['concat']);  
  // $ grunt - 명령어로 기본 task 실행  
  grunt.registerTask('default', ['concat', 'uglify']);  
};
```

Task Runner

Front 도구/개발환경

gulpfile.js

GRUNT / Gulp task 정의 방식 차이

JSON의 선언적 방식으로 테스트를 주입하는 Grunt
(Configuration over programming)

코드의 절차적 방식으로 테스트를 주입하는 Gulp
(Programming over configuration)

JSON에 따른 구조화된 선언

Grunt는 JSON 설정을 기반으로 하고 있기 때문에
모듈화나 구조화를 하더라도
기본적인 선언 형태를 바꾸기 어렵다.

코드에 따른 다양한 방식의 선언

Gulp는 결국 코드이기 때문에
추상화하는 방식이나 모듈화하는 방식에
유연함(다양함)을 줄 수 있어
개발자 성향에 따른 설계가 가능하다.
즉, 각 task 선언 파일을 나누고 복잡해 질수록
동작을 파악하기 어렵다.

```
var pkg = require('./package.json');
var gulp = require('gulp');
var concat = require('gulp-concat'); // 병합
var uglify = require('gulp-uglify'); // 압축/난독화
var rename = require('gulp-rename'); // 파일명 변경

function uglifyTask() {
  return gulp.src(['dist/build/result.js'])
    .pipe(uglify())
    .pipe(rename('result.min.js'))
    .pipe(gulp.dest('dist/build'));
}

function concatTask() {
  return gulp.src('src/js/*.js')
    .pipe(concat('result.js'))
    .pipe(gulp.dest('dist/build'));
}

// $ gulp concat - 명령어로 task 선택 실행
//gulp.task('concat', concatTask);
// $ gulp - 명령어로 기본 task 실행
//gulp.task('default', [concatTask], uglifyTask);

// Gulp 4 방식 (series: 순차실행, parallel: 병렬실행)
exports.default = gulp.series(concatTask, uglifyTask);
```

\$ gulp <task>

Module Bundler

Front 도구/개발환경

2012년 webpack, 모듈 번들러

여러 종속관계에 있는 모듈들을 파악해 하나의 번들파일로 만들어주는 도구 (CommonJS / AMD / ES6 Module / dynamic import)

코드 압축 도구(minifying)를 사용하더라도, 파일단위의 많은 모듈을 불러오게 되면, 네트워크 병목이 발생할 수 있음

단순히 코드를 합쳐주는 도구(concatenating)를 사용할 경우, 합쳐진 파일의 용량이 너무 크면, 여전히 네트워크 지연이 발생할 수 있음

웹팩은 모듈들을 단순히 합치는 것이 아닌, 모듈들의 종속관계를 파악해 사용되는 모듈들만 하나의 번들파일로 생성해 주는 것

웹 페이지에서 초기 구동에 필요한 비동기 방식의 모듈(AMD)들이 많으면,

네트워크 대기시간이 발생할 수 있으나, 번들링을 통해 이를 최소화(사전 미리로드 효과) 할 수 있음

웹 페이지에 넣어 브라우저에서 바로 실행할 수 없는 코드(파일)들을 컴파일해 브라우저에서 실행가능한 형태로 만들수 있음 (Loader)

각 모듈에서 공통으로 사용하는 모듈을 분리(Code Splitting, Dynamic Imports), 해시 등

성능 개선과 최적화(optimization)와 함께, 여러 추가적인 기능(Plugin)들을 옵션형태로 비교적 쉽게 설정할 수 있음

Module Bundler

의존성/종속성을 가진 모듈들을 파악해, 하나의 파일로, 재생산

Task Runner

반복적인 작업을 사전 등록 후, 하나의 명령으로 실행하며, 자동화

Module Bundler

Front 도구/개발환경

webapck.config.js

entry

웹팩이 번들링을 시작하기 위해 필요한 최초 진입점 정의
(종속관계가 시작되는 첫번째 파일지정)

output

웹팩이 번들링한 결과(파일)를 어떤 경로, 어떤 이름,
어떤 형태로 저장할지 정의

loader

웹팩이 번들링의 과정에서, 자바스크립트가 아닌
Scss, Template, Typescript, React, Vue 등의 웹 자원을
실행가능한 형태로 변환해주는 속성
(모듈단위로 파일을 해석하는 과정에서 관여)

plugin

웹팩으로 번들링한 파일에 추가적 기능들을 적용해주는 속성
(번들링 완료 후 마지막 output 과정에 관여)

* 웹팩 4 부터 optimization 최적화 옵션 추가됨

```
var path = require('path');
module.exports = {
  // 엔트리 - 종속관계가 시작되는 첫번째 모듈(진입점) 설정
  entry: {
    'module': './src/javascript/module.js',
  },
  // 로더 - 번들링 과정(모듈단위)에서의 처리할 것 설정
  module: {
    // 웹팩1 loaders 키로 설정, 웹팩2 이상 rules 키로 설정
    rules: [{
      test: /\.handlebars$/,
      loader: "handlebars-loader"
    }]
  },
  // 아웃풋 - 경로, 파일명 등 번들링 결과 설정
  output: {
    // path 는 어디에 웹팩 번들 결과파일을 저장하는지 설정
    path: path.resolve(__dirname, './dist/webpack/'),
    // [name] / [hash] / [chunkhash]
    filename: '[name].bundle.js'
  },
  // 플러그인 - 웹팩 기본기능 외 추가기능 설치 후 사용(부가적인 기능)
  plugins: [

  ],
  // 웹팩4 최적화 관련 플러그인들이 모두 optimization 속성으로 통합
  optimization: {

  }
};
```

\$ webpack

Transcompiler (Transpiler)

Front 도구/개발환경

2015년 babel (6to5 프로젝트), 트랜스 파일러

TypeScript, CoffeeScript, React 등의 언어를 브라우저에서 실행가능한 JavaScript로 변환한 것을 트랜스 파일, 변환해주는 도구를 트랜스 파일러

6to5 라는 이름으로 JavaScript ES6 방식 코드를 ES5 로 바꿔주는 도구로 시작

ES7 이나 다른 트랜스파일도 지원하는 포괄적인 도구가 되기 위해 Babel 이라는 이름으로 변경

예전에는 Babel 을 설치했을 때 많은 기능이 포함되어 있었지만, 6.0 이 되면서 각 기능이 모듈화되어 필요한 것만 선택적 설치로 변경됨

babel은 그 자체로는 아무것도 하지 않음

babel 에서 사용할 preset, plugin 등을 추가(설치)하지 않는다면, babel은 아무것도 하지 않음

(preset 이란 plugin의 집합, 프리셋을 설치할 경우 포함된 플러그인도 자동설치)

즉, babel이 무엇을 하게 하려면 plugin 또는 preset 을 설치해야 함

Babel 설치 (babel-cli는 command line 바벨 실행가능 도구)

```
$ npm install @babel/core @babel/cli
```

Babel plugin 또는 preset 설치

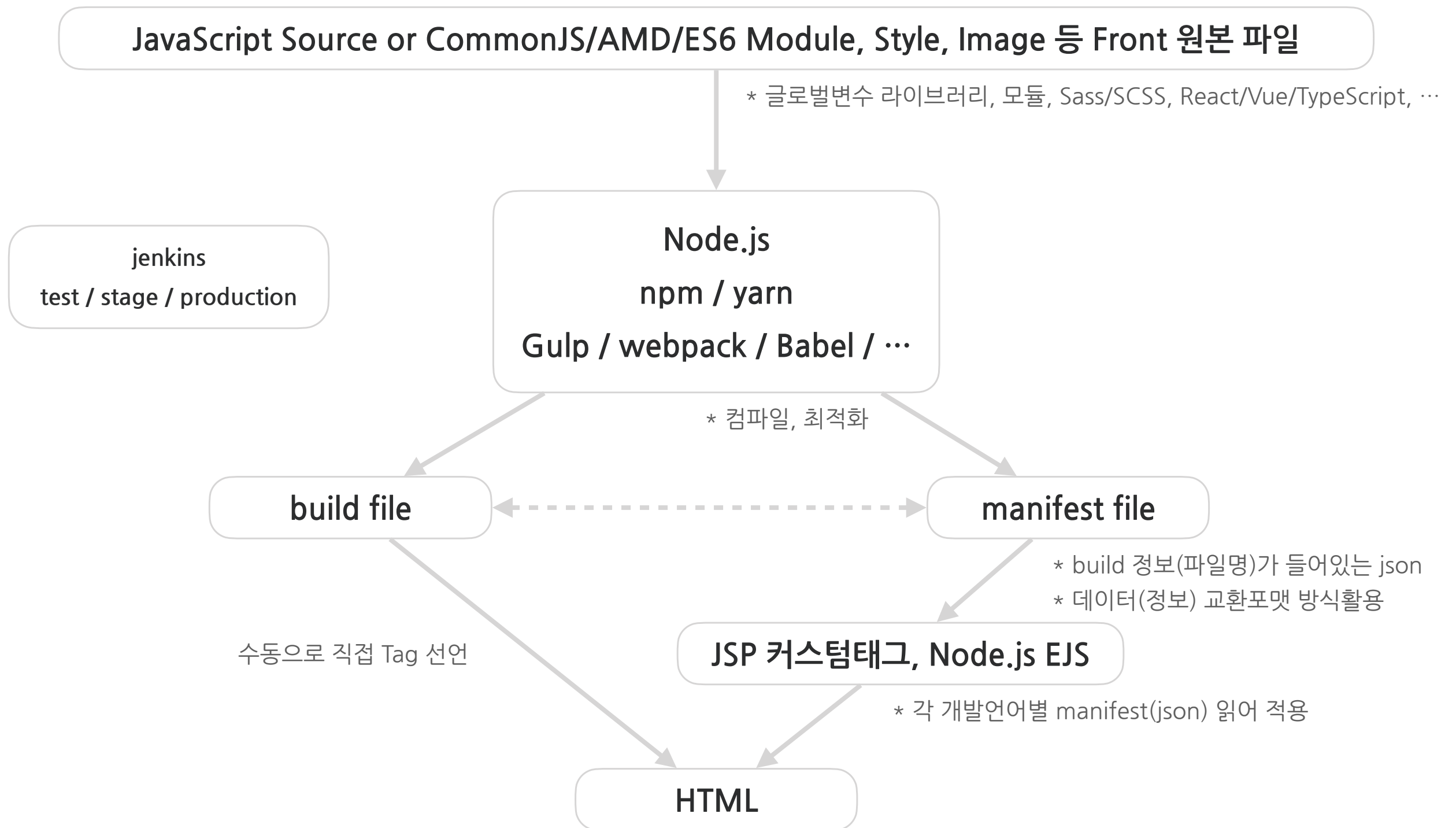
```
$ npm install @babel/preset-env
```

.babelrc

```
{
  presets: ['@babel/preset-env'],
  plugins: []
}
```

바벨에게 어떤 플러그인, 프리셋 등을 사용할 것인지 설정하는 파일

[React 등은 브라우저에서 바로 실행가능한(선택 가능한) 라이브러리 제공]



[2019년, 주관적 관점에서의 Front 개발환경, 자신이 속해있는 개발환경 파악과 그에 따른 문제해결 중심의 도구선택이 더욱 중요하다 생각함]

[새로운 HTTP 표준화 현황, 개발도구 등장에 따라 프론트에서 사용되는 도구도 계속해서 변할 수 있음]

지속적 통합(CI, Continuous Integration)

Front 도구/개발환경

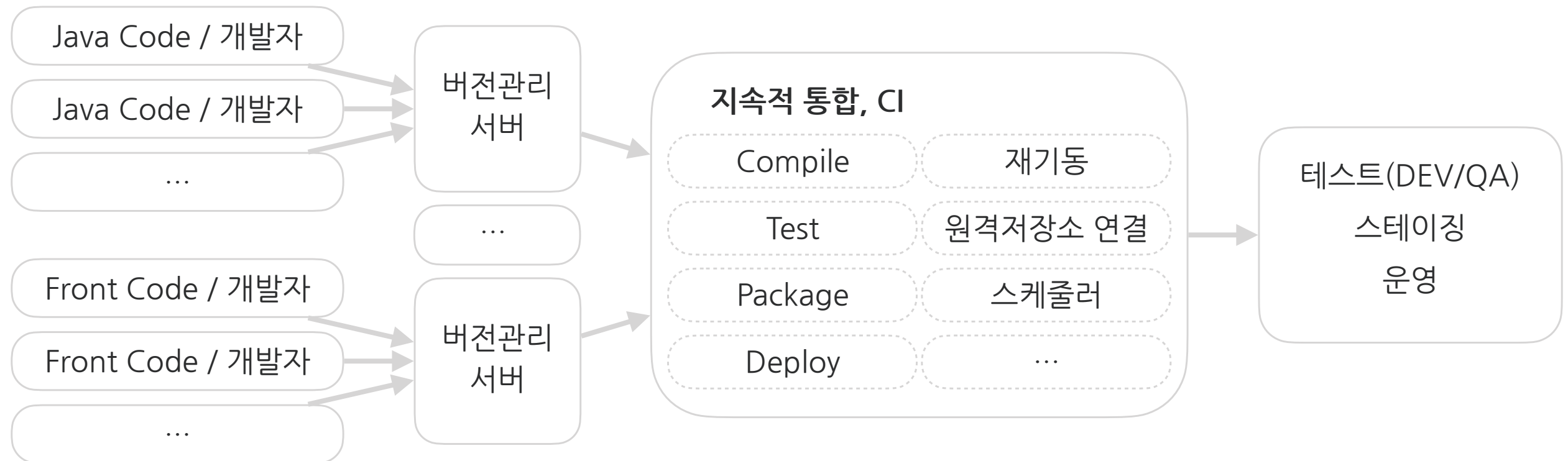
2005년 Jenkins (Hudson Project), 여러 가지 환경들을 통합해서 프로세스화

애플리케이션을 지속적으로 컴파일하고 단위 테스트를 해서, 소프트웨어의 품질을 향상시킬 수 있는 환경을 마련하는 것

개발 프로세스의 각 단계를 자동화해서 얻는 각종 보고서 및 결과물들을 주기적으로 공유하고 확인할 수 있는 환경을 마련하는 것을 목표

짧은 주기로 개발중인 소프트웨어를 확인 가능하도록 배포하고 그 과정을 자동화, 지속적 배포

검증 및 릴리스하는 데 걸리는 시간을 단축



—
감사합니다.