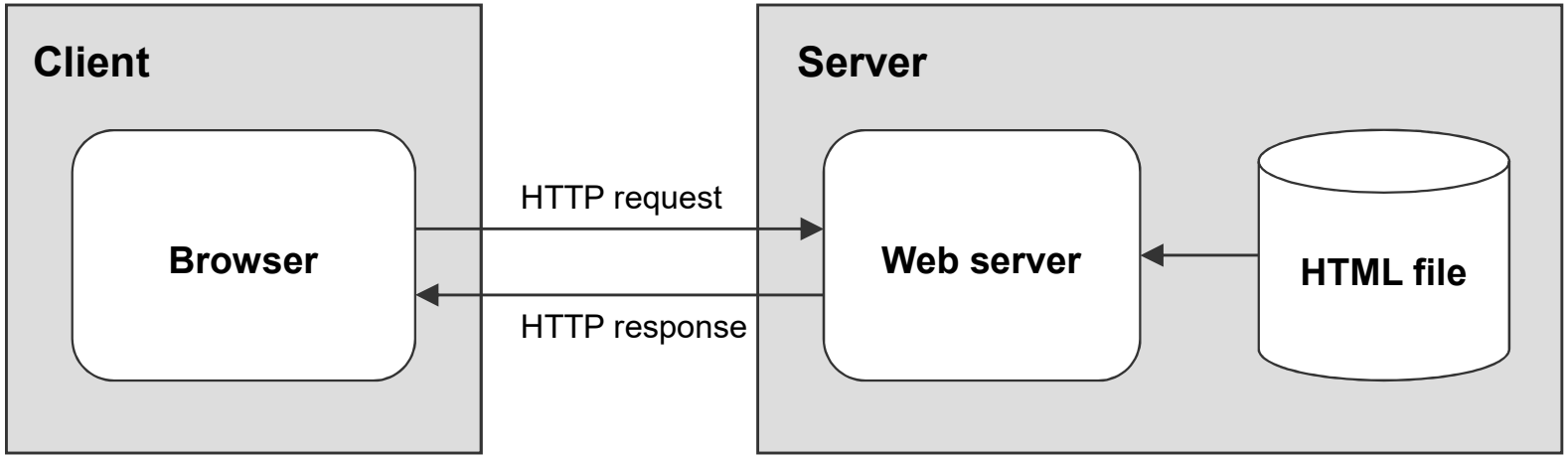
Gulp를 이용한 웹 개발 자동화

JavaScript Task Runner

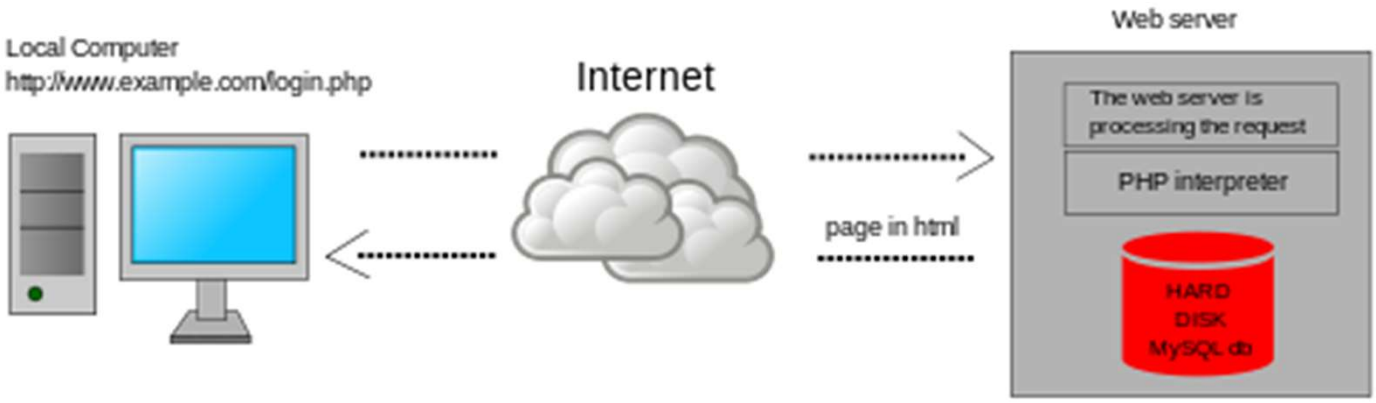
웹이 세상에 등장했을 때
: 정적 웹 페이지

- 클라이언트(브라우저)가 서버상에 있는 파일의 위치를 지정하여 요청
- 서버는 해당 위치에 있는 파일(HTML)을 클라이언트로 전송
- 클라이언트는 HTML을 기반으로 문서의 구조를 렌더링하여 화면에 표시



웹이 활성화되면서
: 동적 웹페이지

- 정적 페이지의 제공만으로 사용자의 다양한 요구를 충족하지 못하게 되면서 서버상에서 데이터를 가공, HTML로 렌더링하여 클라이언트로 전송
- 사용자의 다양한 요구를 반영한 맞춤 정보 제공이 가능해짐
- 사실 이때까지만 해도 Front End 기술은 HTML을 이용, 문서의 구조를 표현하는 수준에 그침



웹 사이트의 표현을 풍부하게 하기 위한 시도
: CSS, JavaScript

- 역할의 분담
  - HTML : 문서의 구조와 데이터를 담당
  - CSS : 문서의 표현 영역(디자인, 스타일)을 담당
  - JavaScript : 문서의 동적인 작동을 담당

JavaScript
: A Brief History

- 넷스케이프가 클라이언트 측에서 사용할 목적으로 개발한 라이브스크립트에서 시작
- 1995년 JavaScript로 명칭을 변경 후 발표
- Microsoft JScript 등 브라우저 개발사별 스크립트의 난립으로 호환성에 문제 발생
- 1997년 넷스케이프가 JavaScript 스펙을 ECMA에 제출, 첫 버전 ECMA-262 발표
- 현재 대부분 브라우저가 ECMA-262를 기반으로 한 JavaScript 1.6, 1.7을 지원
- 2015년 ECMAScript Edition 6(ES6, ES2015) 발표

ECMAScript 2015

- JavaScript 출범 이후 최대의 변화
- JavaScript가 가지고 있던 단점들을 극복하고자 하는 노력
  - 블록 스코프 : let/const
  - 컬렉션 : Map, Set, WeakMap, WeakSet 등
  - class
  - 향상된 for loop
  - Promises
  - Template String
  - 모듈화

- 현실 세계의 문제
  - 실제 브라우저들이 ES2015를 지원하려면 많은 시간이 필요할 것

CSS

- 마크업 언어(HTML)가 실제 표시되는 방법을 기술하는 언어
- 페이지 디자인과 스타일에 관련된 일련의 기술들

- 단순한 문법과 수많은 키워드를 이용한 다양한 스타일의 프로퍼티 이름을 규정

- 현실상의 문제
  - 정적으로 기술하는 문서이므로 사이트가 커짐에 따라 구조적으로 기술하기 쉽지 않고 변경 사항에 유연하게 대처하기 힘들다
  - 대안으로 SASS, LESS 등 객체지향 언어의 특성을 반영한 언어들이 등장

웹 사이트의 고도화

- 웹이 활성화되면서 기존 개발 방식으로는 사용자의 다양한 요구를 반영하지 못함
- 인터넷이 가진 장점을 활용하면서도 구조적, 효율적으로 Front End를 개발해야 할 필요성이 대두 -> Front-End Engineering 개념이 도입
  - 단순 웹 페이지 출력이 아닌 고도화된 MVC 형태의 개발
  - 페이지 전환 없는 데스크탑 응용 프로그램 수준의 웹 어플리케이션 구현

- 풍부한 UX와 안정적인 웹 어플리케이션의 작동을 구현하기 위한 다양한 시도들
  - 각종 Front-End Framework, Transpiler 들이 등장

Transpiler (혹은 Preprocessor)

- CSS를 확장하기 위한 언어들 : SASS, LESS, Stylus 등
- JavaScript를 확장하기 위한 언어들 : CoffeeScript, TypeScript 등

- 하지만 이런 언어들은 실제 브라우저에서는 작동하지 않음
- 이러한 언어들로 만들어진 소스를 CSS, JavaScript 소스로 변환하는 작업이 필요
- 이 변환 작업을 담당하는 소프트웨어를 트랜스파일러(혹은 크로스 컴파일러)라 함

- 컴파일러와의 차이
  - 컴파일러 : 소스 -> 실행 가능한 기계어로 변환
  - 트랜스파일러 : 소스 -> 다른 언어의 소스로 변환

난립하는 솔루션과 프레임워크

- 좋기는 한데... 해야 할 일이 늘었다

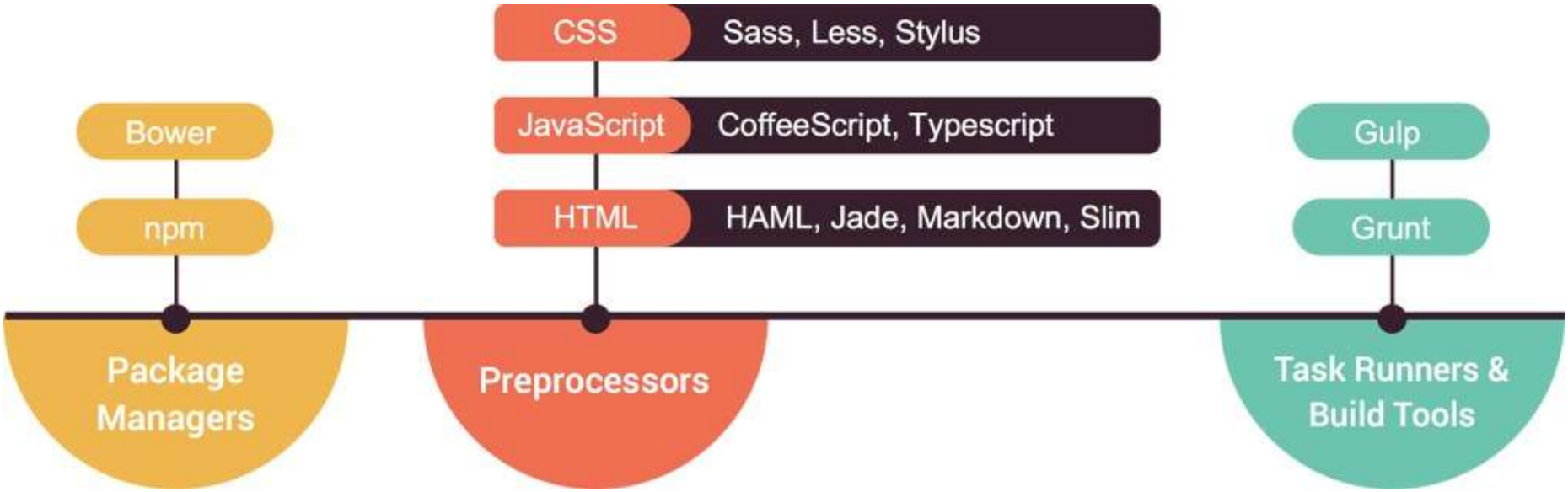
늘어난 Front-End Engineer의 일

- Transpiling
  - SASS, LESS -> CSS
  - CoffeeScript, TypeScript -> JavaScript
- Concatenation
  - 분할해서 개발하는 소스를 하나로 합치는 일
- Image Optimization
  - jpg, png 등 이미지를 최적화하여 트래픽을 줄이는 일
- Source Compression(혹은 Minification)
  - JavaScript, CSS, HTML 등에서 필요 없는 공백 등을 제거, 사이즈를 줄이는 일 혹은 코드 난독화
- 변경된 소스를 실제 브라우저에서 확인

- 이 일들은 대부분 반복적이고 지루한 작업들…
  - 효율적으로 일하고 싶어요…

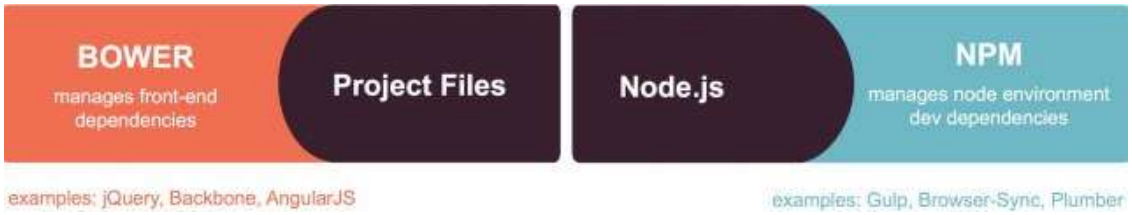
JavaScript 기반 Build System

- 단순 반복되는 작업을 자동화
  - Transpiling
  - Concatenation
  - Minification
  - 브라우저 자동 리로딩 등



JavaScript 기반 Build System
: Package Manager

- 개발 환경에서 사용되는 라이브러리, 의존성 패키지 등을 설치, 업데이트, 삭제하는 부분을 자동화
  - Bower
  - NPM (Node.js)



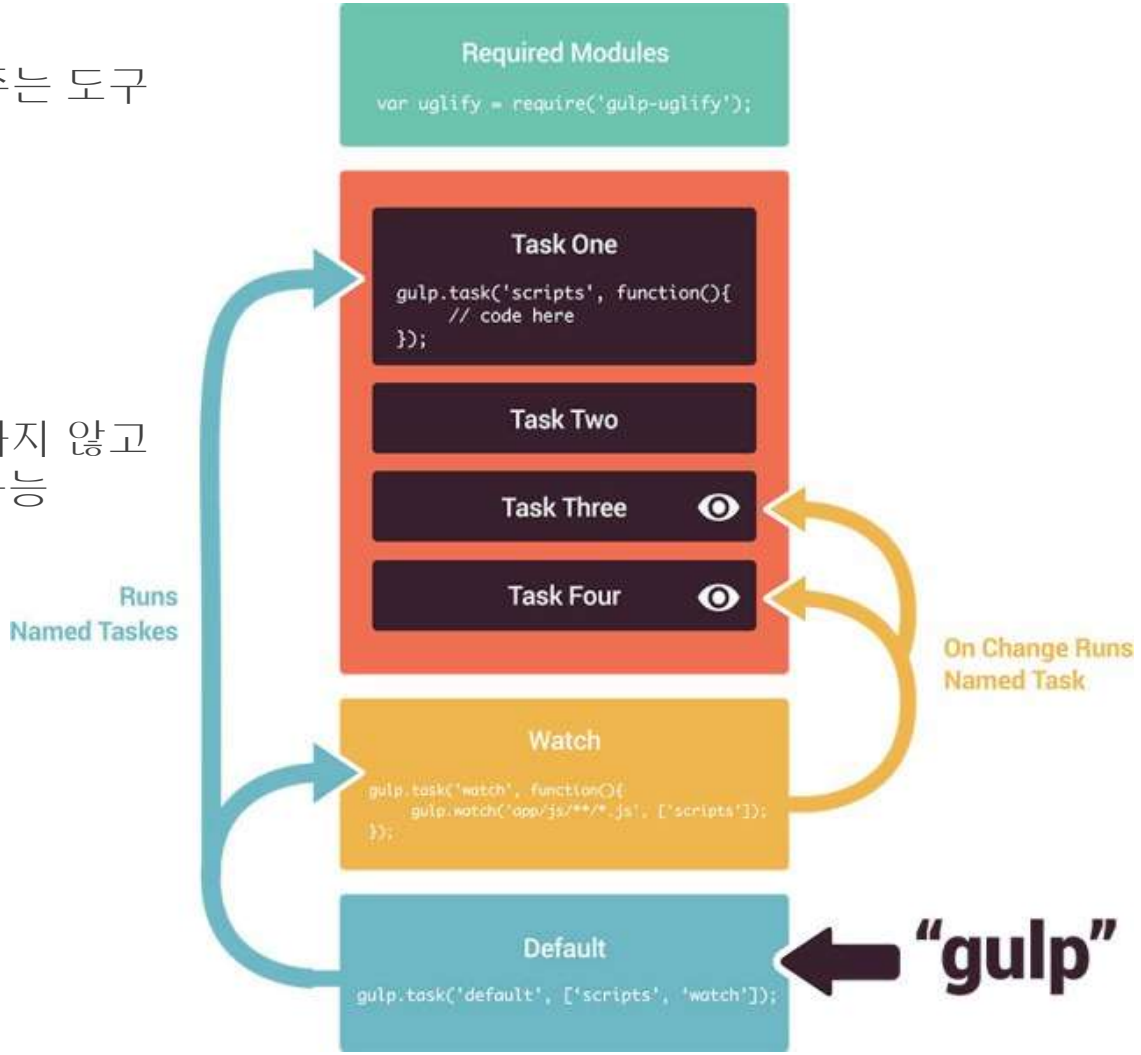
JavaScript 기반 Build System
: Transpiler(Preprocessor)

- 추가 기능 혹은 최적화된 문법을 제공하는 언어를 기반이 되는 원래 언어로 변환해 주는 소프트웨어
- 웹 기반 언어의 전처리기들
  - CSS : SASS, LESS, Stylus 등
  - JavaScript : CoffeeScript, TypeScript 등
  - HTML : HAML, Jade, Markdown, Slim 등

JavaScript 기반 Build System
: Task Runner

- 반복되는 일련의 작업들을 자동으로 수행해주는 도구
- 유명한 Task Runner
  - Gulp
  - Grunt

- Grunt가 원조격이지만, Gulp는 설정이 복잡하지 않고 JavaScript 코드로 태스크를 설정하여 사용 가능



Gulp

Gulp 소개

- Node.js를 플랫폼으로 하는 Task Runner
- 순수 JavaScript 코드로 작성하며 Front-End 작업과 방대한 규모의 웹 응용프로그램 작성을 도와줌

- 2015년 1월 15일 v1.0.0 발표, 현재 v3.9.0

- 주요 기능
  - 코드 최적화(minification)와 concatenation(합치기)
  - 순수 자바스크립트 코드
  - LESS/SASS to CSS Transpilation
  - Node.js 플랫폼을 활용, 메모리에서 파일 조작을 관리하고 속도를 향상시킴

Gulp 프로젝트 설정
: Gulp 설치

- gulp 설치

```
npm install -g gulp
```

- 노드 프로젝트 init

```
npm init
```

- Local gulp 설치

```
npm install gulp --save-dev
```

Gulp 프로젝트 설정
: gulpfile.js 작성

- Gulp는 실행시 gulpfile.js 내용을 참조, Task를 실행

```
var gulp = require('gulp');
```

- 태스크의 작성

```
gulp.task('mytask', function() {
    console.log("My First Gulp Task");
})
```

태스크 작성법

```
gulp.task('task-name', function() {
    // 태스크에 실행할 내용
});
```

- Task의 실행

```
gulp mytask
```

Gulp의 주요 메서드

메서드	문법	설명
task	.task(string, function)	태스크의 이름과 해당 태스크의 실행 함수
src	.src(string || array)	소스파일(혹은 입력)에 대한 정보
pipe	.pipe(function)	단일 목적의 함수 혹은 플러그인을 연쇄 파이프로 연결
desc	.desc(string)	출력파일(혹은 출력)에 대한 정보
watch	.watch(string, array)	파일이 변경되었는지 감시하기 위한 메서드

Gulp 응용
: 이미지 최적화

- gulp-imagemin 플러그인 설치

```
npm install gulp-imagemin --save-dev
```

- Task의 작성

```
var imagemin = require('gulp-imagemin');

gulp.task('images', function() {
    var imgsrc = 'src/img/**/*';
    var imgdest = 'dist/img';
    return gulp.src(imgsrc)
        .pipe(imagemin())
        .pipe(gulp.dest(imgdest));
});
```

Gulp 응용
: CSS 합치기 + Clean

- gulp-concat, gulp-clean-css, gulp-myth 플러그인 설치

```
npm install gulp-concat gulp-clean-css gulp-myth --save-dev
```

- Task의 작성

```
var concat = require('gulp-concat');
var myth = require('gulp-myth');
var cleanCss = require('gulp-clean-css');

gulp.task('styles', function() {
    return gulp.src('src/css/**/*')
        .pipe(concat('site.css'))
        .pipe(myth())
        .pipe(cleanCss({compatibility:'ie8'}))
        .pipe(gulp.dest('dist/css'));
});
```

Gulp 응용
: JavaScript Transpilation + 합치기 + 코드 난독화

- gulp-babel, gulp-uglify 플러그인 설치

```
npm install gulp-babel gulp-uglify --save-dev
```

- babel-core, babel-preset-es2015 설치

```
npm install babel-core babel-preset-es2015 --save-dev
```

- Task의 작성

```
var babel = require('gulp-babel');
var uglify = require('gulp-uglify');

gulp.task('scripts', function() {
    return gulp.src('src/js/**/*.js')
        .pipe(concat('site.js'))
        .pipe(babel({
            "presets": "es2015"
        }))
        .pipe(uglify())
        .pipe(gulp.dest("dist/js"));
});
```

Gulp 응용
: 여러 태스크를 한데 모아 실행

- Task 작성시 여러 태스크를 한데 모아 실행할 수 있다

```
gulp.task("multitask", ['styles', 'scripts', 'images']);
```

- Task 이름을 default로 부여하면 태스크명 없이 태스크를 실행할 수 있다

```
gulp.task("default", ['styles', 'scripts', 'images']);
```

- 기본 태스크의 실행

```
gulp
```

Gulp 응용
: 파일 변경 감지

- gulp.watch 메서드를 이용하면 파일이 변경되는 사항을 감지할 수 있다

```
gulp.task("watchdog", function() {
    gulp.watch('src/css/**/*.css', ['styles']);
    gulp.watch('src/js/**/*', ['scripts']);
});
```

- 파일 감시를 시작해 봅시다

```
gulp watchdog
```

- 소스 파일을 변경하여 파일 감시가 진행되는지 확인

Gulp 응용
: Browser Sync를 이용한 변경사항 자동 리로딩

- browser-sync 설치

```
npm install browser-sync --save-dev
```

- Task의 작성

```
var browserSync = require('browser-sync');

gulp.task('browserSync', function(cb) {
    return browserSync({
        server: {
            baseDir: "./"
        },
        browser: "chrome"
    }, cb);
});
```

Gulp 응용
: Browser Sync를 이용한 변경사항 자동 리로딩

- 소스가 변경될 때 브라우저를 리로드해 봅시다(watchdog 태스크를 재정의)

```
gulp.task("watchdog", function() {
    gulp.watch('src/css/**/*.css', ['styles', browserSync.reload]);
    gulp.watch('src/js/**/*', ['scripts', browserSync.reload]);
    gulp.watch('*.html', browserSync.reload);
});
```

- default Task에 모두 모아 실행해 봅시다

```
gulp.task("default", ['styles', 'scripts', 'images', 'browserSync', 'watchdog']);
```

- HTML, CSS, JavaScript 등을 변경해 보고 브라우저가 리로드되는지 확인

어려운 일에 관해서 난 항상 게으른 사람을 선택한다.
그들은 가장 쉬운 방법을 찾아낼 테니까.
- Bill Gates