



softcontext@gmail.com

## 내용

Chapter 1 : ECMAScript.....	5
1. ECMAScript .....	6
2. ES6 개발환경 설정.....	7
WebStorm 바벨 설정.....	7
3. var .....	10
4. let .....	13
5. const .....	14
6. parameter.....	15
7. spread operator.....	17
8. Destructuring Assignment.....	19
9. Module System.....	20
1. CommonJS 방식 .....	21
2. ES6(ECMA2015) 방식 .....	23
3. IIFE 방식.....	34
4. AMD .....	35
10. 함수 축약 표현식 .....	38
11. Arrow function.....	39
12. \$ expression.....	48
13. class.....	49
클래스 주요 특징 .....	49
클래스 기본 문법 .....	49
클래스 문법 코드를 ES5 코드로 트랜스파일링한 결과 확인 .....	51
extends 키워드 .....	53
ES5, ES6 문법 혼용 .....	54
클래스가 객체를 대상으로 상속 .....	55
super 키워드 .....	56
클래스 표현식 vs 선언식 .....	58
ES5 상속과 class 를 사용한 ES6 상속의 차이점.....	60
Multiple Inheritance with Proxies.....	62
14. 비동기 처리 .....	63
1. Call-back Function .....	63
2. Event Emitter .....	67
3. Promise .....	71

15. TypeScript 소개.....	76
Learn TypeScript in 30 Minutes.....	76
The Benefits of Using TypeScript.....	76
Chapter 2 : Angular Basic.....	77
Step 1 – 앵귤러 소개 .....	78
Step 2 – Simple Example .....	81
Step 3 – Project Structure .....	82
Step 4 - CLI .....	88
1. 설치.....	88
2. 새 프로젝트 만들기.....	93
3. 테스트를 위한 빌드.....	99
4. 서비스(배포)를 위한 빌드 .....	101
Step 5 – Component.....	103
Binding.....	103
빌트인 지시자 .....	119
Step 6 – Service.....	137
서비스 추가 .....	140
Step 7 – Pipe .....	143
빌트인 파이프 .....	143
커스텀 파이프 .....	148
Step 8 - Directive .....	154
커스텀 디렉티브.....	154
Step 9 – Module .....	164
루트 모듈 : AppModule .....	165
핵심 모듈 : CoreModule .....	173
특징 모듈 : PlayerModule .....	177
특징 모듈 : MemberModule .....	179
공유 모듈 : ShareModule .....	184
Chapter 3 : Angular Core .....	191
Step 1 – Life Cycle.....	192
Step 2 – Component Communication .....	206
@Input and @Output.....	206
@ViewChild.....	210
Observable & Subject .....	215

@ContentChild.....	223
@ViewChildren.....	227
@ContentChildren.....	232
Step 3 – HTTP .....	237
Promise .....	237
Observable .....	243
Step 4 – Router.....	257
해시 기반 주소로 변경 .....	262
연결순서 : http://localhost:4200/router-link-test .....	263
연결순서: http://localhost:4200/pages/first-page .....	265
연결순서 : http://localhost:4200/member .....	268
연결순서 : http://localhost:4200/children.....	270
연결순서 : http://localhost:4200/login?session_id=1234#anchor.....	276
연결순서 : http://localhost:4200/children.....	279
연결순서 : http://localhost:4200/children/1 .....	279
연결순서 : /children/1 → /children/5 .....	283
연결순서 : http://localhost:4200/lazy/player.....	286
연결순서 : http://localhost:4200/active.....	288
특징 모듈 라우터 .....	291
Step 5 - Guard.....	294
Step 6 – Form .....	297
book-form-basic .....	297
book-form-valid .....	299
book-form-control .....	301
book-form-formbuilder .....	304
FormArray .....	306
Chapter 4 : Angular Extension.....	307
Step 1 – DI.....	308
Providers.....	310
불투명 토큰을 이용한 제공자 설정 .....	328
Provider 없이 객체 DI.....	330
주입기를 이용한 객체 생성 .....	332
Step 2 – CSS Style .....	336
Step 3 – Sanitization .....	342

Step 4 – Animation .....	346
Chapter 5 : Angular Deep Dive .....	357
Step 1 - Angular1 과의 차이점 .....	358
Directive.....	373
Components VS Directive .....	373
Step 2 – 분석 : Registration and Login.....	374
Step 3 - 실습 : Tour of Heroes Tutorial.....	375
01 : The Hero Editor .....	376
02 : Master/Detail.....	382
03 : Multiple Components.....	391
04 : Services.....	396
05 : Routing.....	402
06 : Http .....	422
Chapter 6 : Etc.....	449
Transclusion using ng-content.....	450
Shadow DOM .....	458
Light DOM.....	458
ViewEncapsulation .....	459
_nghost-*, _ngcontent-* .....	459
ViewEncapsulation.Emulated .....	461
ViewEncapsulation.Native .....	463
ViewEncapsulation.None.....	467



## Chapter 1 : ECMAScript

앵귤러는 러닝커브가 높은 기술이다. 자바스크립트의 기초가 없는 상태에서 앵귤러를 학습하는 것은 비효율적이다. 학습자가 어려움에 처하지 않고 부드럽게 앵귤러의 학습이라는 문으로 들어갈 수 있도록 이번 장에서 자바스크립트의 기초를 다지기 위해서 새로 도입된 문법을 먼저 익힌다. 더불어서, 앵귤러 코드를 짤 때 사용하는 타입스크립트에 대해서 학습한다.

- ECMAScript 소개
- 개발환경 설정
- ES6, 7에서 추가된 문법
- 모듈시스템
- 바벨을 이용한 트랜스파일링
- 클래스
- 비동기 처리 기술
- TypeScript 소개

# 1. ECMAScript

ES6는 ECMAScript 6의 줄임말이다. ECMAScript 2015라고도 부른다. 따라서, ES7은 ECMAScript 2016이 된다. ECMAS는 European Computer Manufacturers Association의 약자이다. ECMA인터넷은 정보와 통신 시스템을 위한 국제적인 회원국 기반의 비영리 표준화 기구이다.

아직까지 대부분의 브라우저는 최신 문법을 지원하지 않는다. 그럼에도 불구하고 많은 개발자들이 ES6, 7의 문법을 사용하여 개발할 수 있는 이유는 트랜스파일링 도구 덕분이다. 새 문법으로 작성된 코드를 트랜스파일링 작업을 통해 현 브라우저가 지원하는 ES5 문법으로 변경하여 배포한다. 인기있는 트랜스파일링 도구인 바벨(구 6to5)은 ES6 문법을 ES5 호환 JS 파일로 변환해 주는 대표적인 기술이다. 따라서, 최근에 인기있는 리액트, 앵귤러 등과 같은 프레임워크에서 ES6 문법을 사용하여 개발할 수 있다.



## ES6 브라우저 지원현황

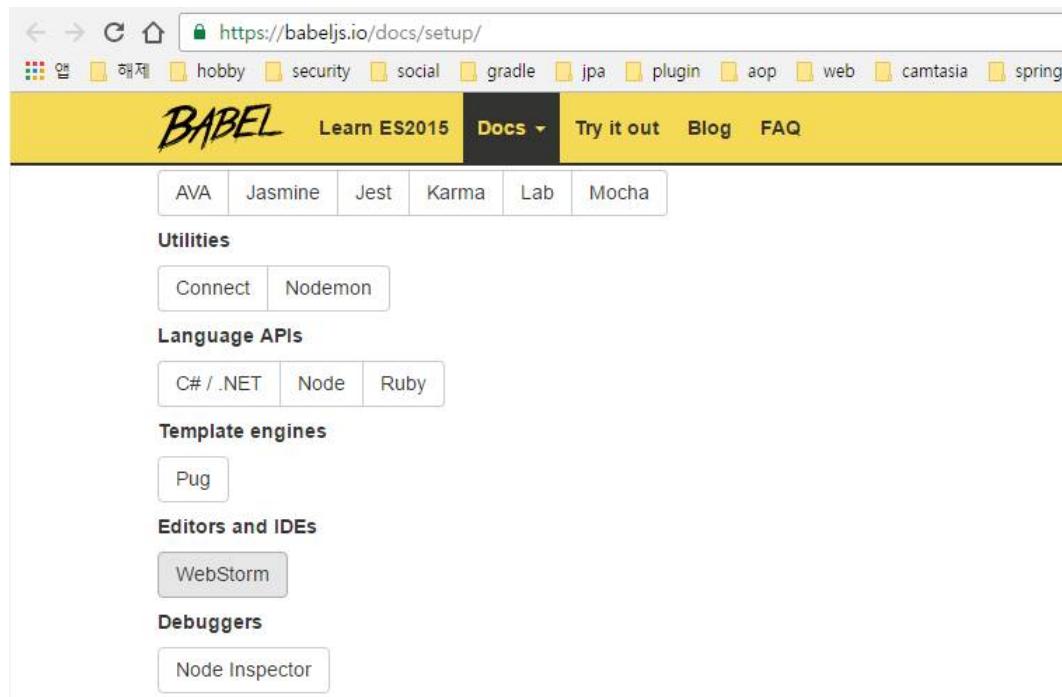
<https://kangax.github.io/compat-table/es6/>

## 2. ES6 개발환경 설정

- Node 설치 : JS의 단독 실행환경 + 기본 라이브러리(fs, http 모듈 등)
- Git 설치 : Git CMD, Git Bash(MinGW 수준의 리눅스 명령어 사용가능)
- WebStorm 설치 : IDE Tool

## WebStorm 바벨 설정

참고: <https://babeljs.io/docs/setup/>



"WebStorm" 버튼을 클릭하면 바벨 설정안내 페이지로 이동한다.

## Installation

```
npm install --save-dev babel-cli
```

## Usage

In **Preferences - Tools - File watchers**, click **+ button** and select **Babel file watcher** from the list.

Specify the path to Babel executable and click Ok.

By default all files with a **.js** extension will be automatically compiled with Babel upon change. The generated ES5 files and source maps will be saved next to original files.

Lastly, in **Languages & Frameworks - JavaScript - JavaScript language version**, choose **ECMAScript 6**.

## Create **.babelrc** configuration file

Great! You've configured Babel but you haven't made it actually do anything. Create a **.babelrc** config in your project root and enable some plugins.

To start, you can use the the latest preset, which enables transforms for ES2015+

```
npm install babel-preset-latest --save-dev
```

In order to enable the preset you have to define it in your **.babelrc** file, like this:

```
{
  "presets": ["latest"]
}
```

Note: Running a Babel 6.x project using npm 2.x can cause performance problems because of the way npm 2.x installs dependencies. This problem can be eliminated by either switching to npm 3.x or running npm 2.x with the dedupe flag. To check what version of npm you have run

```
npm --version
```

## 바벨 트랜스파일링

콘솔에서 다음 명령을 사용하여 트랜스파일링 직접할 수 있다.

대상 ES6.js를 트랜스파일링한 후 디렉토리 build 밑에 생성하는 명령이다.

```
babel --presets latest ES6.js -d build
```

### 3. var

변수 선언 시 사용하는 키워드인 var 연산자의 문제점을 파악해 보자.

#### var1.js

```
// var 키워드로 선언한 변수는 function scoped variable 이다.  
// 유독 JS 만이 블록연산자 스코프 변수라는 개념이 없었다.  
  
var a = 1; // 전역 접근 가능  
console.log('a = '+a);  
  
(function some(){  
    console.log('a = '+a);  
  
    var b = 2; // 함수안에서 접근 가능  
    console.log('b = '+b);  
  
    if(true){  
        console.log('b = '+b);  
  
        var c = 3;  
        console.log('c = '+c);  
    }  
  
    console.log('c = '+c); // 접근 가능!  
})();  
  
// console.log('b = '+b); // 에러  
// console.log('c = '+c); // 에러  
  
// #출력결과  
// a = 1  
// a = 1  
// b = 2  
// b = 2  
// c = 3  
// c = 3
```

## var2.js

```
// if, for 블록연산자는 var 키워드로 선언된 변수의 스코프를 제한하지 못했다.  
// 함수의 블록연산자만이 var 키워드로 선언된 변수의 스코프를 제한한다.  
  
for(var i=1; i <= 3; i++){  
    console.log(i = '+i);  
}  
  
console.log(i = '+i); // 접근 가능!  
  
// #출력결과  
// i = 1  
// i = 2  
// i = 3  
// i = 4
```

## var3.js

```
var a = 1;  
var a = 2; // 심지어 재선언도 가능하다!  
  
console.log(a = '+a);  
  
let b = 1;  
// let b = 2; // 에러가 발생한다. 변수 재선언이 불가능하다!
```

## var4.js

---

```
var a = 1;
let b = 1;

(function some() {
    var a = 2; // 다른 변수
    let b = 2; // 다른 변수

    if(true){
        var a = 3; // 변수 재선언!
        let b = 3; // 다른 변수
        console.log('a = '+a); // 3
        console.log('b = '+b); // 3
    }

    console.log('a = '+a); // 3 (재선언 변수를 사용)
    console.log('b = '+b); // 2
})();

console.log('a = '+a); // 1
console.log('b = '+b); // 1
```

---

## 4. let

변수 선언 시 사용하던 var 키워드는 조건문의 범위연산자를 무시하여 버그를 양산하던 주범이었다.

변수 선언 시 let 키워드를 사용하면 다른 언어들과 마찬가지로 철저하게 범위 연산자를 "{}"에 따라 변수의 스코프가 결정된다. 따라서 자바의 범위연산자의 스코프와 같다고 생각해도 된다.

### let1.js

```
// let 키워드로 변수를 선언하면 범위연산자에 따른 스코프를 갖는다.  
// 이제 JS 도 변수의 스코프가 다른 언어와 같아졌다.  
  
let a = 1; // 전역에서 접근 가능  
console.log('a = '+a); // 1  
  
(function some(){  
    console.log('a = '+a); // 1  
  
    let b = 2; // 함수안에서 접근 가능  
    console.log('b = '+b); // 2  
  
    if(true){  
        console.log('b = '+b); // 2  
  
        let c = 3;  
        console.log('c = '+c); // 3  
    }  
  
    // console.log('c = '+c); // 에러가 발생한다. 접근이 불가능하다!  
})();
```

### let2.js

```
// 모든 블록연산자는 let 키워드로 선언된 변수의 스코프를 제한한다.  
  
for(let i=1; i < 3; i++){  
    console.log('i = '+i);  
}  
  
// console.log('i = '+i); // 에러가 발생한다. 접근이 불가능하다!
```

## 5. const

드디어 자바스크립트에서도 상수를 사용할 수 있게 되었다.

최초 할당 이후에는 값의 변경이 허용되지 않는다.

### const.js

```
var PI = 3.141592;
console.log('PI = '+PI);

PI = 3.14;
console.log('PI = '+PI);

// 변수는 대소문자를 구분한다.
const pi = 3.141592;
console.log('pi = '+pi);

// pi = 3.14; // error!

// -----
// # 상수 변수에 객체를 할당
const a = {name:'Chris'};
console.log('a = '+JSON.stringify(a));

a.name = 'Aaron';
console.log('a = '+JSON.stringify(a));

// a 는 객체의 주소(참조값)를 가리킨다.
// a 는 상수로 참조값은 변하지 않지만 객체의 프로퍼티는 변경할 수 있다.
```

## 6. parameter

자바스크립트에서는 함수 호출은 함수명만으로 연동된다.

함수가 받는 파라미터는 옵션으로써 명시해도 되고 선언하지 않아도 된다.

이에 따라 많은 개발자들이 파라미터를 생략하곤 했었다. 심지어 라이브러리 개발자들도 여기에 포함된다.

문제는 라이브러리 함수를 이용하는 컨슈머 개발자들의 입장에서는 파라미터의 생략이 불편함을 초래한다는 점이다. 대부분의 개발자들이 IDE 툴을 사용하여 개발을 한다. 그런데 그 동안 IDE 툴들이 자동완성 기능을 제공하지 못했었다. 함수 선언 시 파라미터가 선언되어 있지 않으면 아무리 좋은 IDE 툴이라 하더라도 도움말 및 자동완성 기능을 제공할 수가 없기 때문이다.

따라서 다른 개발자들이 이용하는 함수를 만드는 개발자는 반드시 파라미터를 명시하는 것이 좋다라는 결론에 다다르게 된다. 이에 따라 파라미터를 명시하는 타입스크립트가 등장하게 되었다.

다음 예제에서 만약 파라미터를 주지 않고 함수를 호출하는 경우 편리하게 초기값을 설정해서 사용할 수 있는 새로운 문법을 살펴본다.

### parameter.js

```
function add(x, y){  
    return x + y;  
}  
// 함수가 파라미터를 못 받는 경우 대신 기본값을 설정할 필요가 있다  
console.log(add()); // NaN  
  
// 연산자 // 를 사용하여 받은 파라미터가 없는 경우 대신 기본값을 할당한다.  
function add2(x, y){  
    x = x || 0;  
    y = y || 0;  
    return x + y;  
}  
console.log(add2()); // 0  
  
// 새로운 문법으로 파라미터를 받지 못하면 대신 기본값을 사용하도록 설정한다.  
function add3(x = 0, y = 0){  
    return x + y;  
}  
console.log(add3()); // 0  
  
// 심지어 파라미터 자리에서 간단한 연산도 된다!
```

```
function add4(x = 0, y = 0, z = x+y){  
    return z;  
}  
console.log(add4()); // 0  
console.log(add4(1, 2)); // 3  
console.log(add4(1, 2, 999)); // 999
```

---

## 7. spread operator

스프레드 연산자 또는 펼침 연산자라고 부른다. 배열의 요소를 하나씩 나누어 처리할 수 있는 편리한 문법이다.

### spread.js

```
// 함수는 파라미터를 낱개로 받는다.  
  
function add(a, b) {  
    return a + b;  
}  
  
  
// 마침 갖고 있는 데이터의 자료형이 배열이다.  
let data = [1, 2];  
  
  
// 배열 요소를 직접 하나씩 꺼내서 파라미터로 전달한다. 불편하다!  
console.log(add(data[0], data[1])); // 3  
  
  
// apply 메소드의 두번째 파라미터로 배열을 전달하면서 함수 add를 호출한다.  
  
// 배열의 아이템들이 많을수록 이전 방법 보다는 편리하다.  
console.log(add.apply(null, data)); // 3  
  
  
// 스프레드 연산자 ...를 사용한다. 배열의 아이템들을 꺼내고 펼쳐서 전달한다.  
// 1, 2 로 치환한 다음 add 함수를 호출한다.  
console.log(add(...data)); // 3  
  
console.log('-----');  
  
// #다른 사용예  
  
let array1 = [2, 3, 4];  
  
  
// 배열 array1을 분해해서 array2에 넣는다.  
let array2 = [1, ...array1, 5];  
  
console.log(array2); // [ 1, 2, 3, 4, 5 ]  
  
console.log('-----');  
  
  
// #다른 사용예  
  
array1 = [1, 2];  
array2 = [3, 4];
```

```

// 두 배열을 붙인다.
let array3 = array1.concat(array2);

console.log(array1); // [ 1, 2 ]
console.log(array2); // [ 3, 4 ]
console.log(array3); // [ 1, 2, 3, 4 ]

// push 함수를 사용하면 array2가 가리키는 배열이 통째로 들어간다.
array1.push(array2);
console.log(array1); // [ 1, 2, [ 3, 4 ] ]

// reset
array1 = [1, 2];
array2 = [3, 4];

// array2 배열의 아이템을 꺼내고 펼쳐서 array1 배열에 추가한다.
array1.push(...array2);
console.log(array1); // [ 1, 2, 3, 4 ]

console.log('-----');

// #rest parameter
// 파라미터 자리에 쓰이면 스프레드 연산자란 용어 대신
// 레스트 파라미터라고 부른다.

// 세 번째 파라미터는 사용되지 않는다.
function subtract(a, b) {
    return a - b;
}

// 전달 받은 파라미터 중 맨 앞에 하나를 변수 a에 할당한다.
// 다른 파라미터는 레스트 파라미터 문법으로 설정하여 args 배열이 갖고 있다.
function calc(a, ...args){
    console.log(Array.isArray(args)); // true
    console.log(args); // [ 1, 2, 3 ]

    switch(a){
        case '+':
            return add(...args);
        case '-':
            return subtract(...args);
        default:
            return 0;
    }
}

console.log(calc('+', 1, 2, 3)); // 3
console.log(calc('-', 1, 2, 3)); // -1

```

## 8. Destructuring Assignment

해체할당이라고 번역해서 부른다. 객체의 일부 프로퍼티만을 사용하고자 할 때 유용한 문법이다.

### destructuring-assignment.js

```
var log = console.log;

var myObj = {a: 1, b: 2, c: 3};

// 객체.프로퍼티 문법으로 접근해서 하나씩 할당한다. 불편하다!
var aa = myObj.a;
var bb = myObj.b;

log('aa = ' + aa); // 1
log('bb = ' + bb); // 2

log('-----');

// 객체는 프로퍼티명을 사용하여 해체할당 문법을 사용할 수 있다.

// 사용된 변수 a, c는 반드시 객체의 프로퍼티명과 같아야 한다.
var {a, c} = myObj;

log('a = ' + a); // 1
log('c = ' + c); // 3

log('-----');

var array = [1, 2, 3];

// 배열은 인덱스기반으로 해체 할당할 수 있다.
var [x1, , x3] = array;

log('x1 = ' + x1); // 1
log('x3 = ' + x3); // 3
```

## 9. Module System

옛날 옛적에 호랑이 담배피던 시절에는 JS 코드를 담고 있는 파일을 다른 JS 파일에서 사용할 수 없었다.

어처구니 없지만 사실이다! 하는 수 없이 HTML에 도움을 받아 결합해서 사용해야만 했다. 차차 이 치명적인 문제점을 해결하기 위한 여러 모듈시스템이 등장하기 시작했다.

HTML 파일에서 .js 확장자의 파일들을 임포트하면 JS 파일들은 하나의 파일처럼 취급된다. 이 방법에 문제점은 파일이 많아지거나 여러 개발자들이 나누어서 개발한다면 변수의 충돌이 빈번하다는 점이다.

이에 따라 글로벌 스코프 변수를 사용하는 것을 금기가 되었고 스코프를 완벽하게 분리하는 방법이 필요했다.

불편함을 느낀 많은 개발자들이 앞 다투어 자바스크립트의 치명적인 약점을 개선하고자 시도했다. 이에 따라 JS 코드를 여러 파일로 분리해서 개발하고 조립하여 사용하는 모듈시스템들이 속속 발표되었다.

자바스크립트에게 자유를 안겨 준 노드는 CommonJS 방식의 모듈시스템을 사용한다.

CommonJS는 스펙이고 구현체는 require.js 라이브러리이다.

한편 표준 단체인 ECMA는 ES6에서 export, import로 대표되는 모듈시스템을 지원하기 시작했다.

노드를 사용하여 자바스크립트를 실행하는 서버환경이라면 require 문법을 사용하는 것이 좋다.

문제는 클라이언트 사이드인데 앱글러, 리액트 모두 ECMA의 import 문법을 선호해서 발생한다.

import 문법을 노드는 지원하지 않기 때문이다. 따라서 6to5라고 불렸던 Babel을 통해 별도로 트랜스파일링을 진행해야 노드 기반에서 코드를 실행할 수 있다.

대부분의 자바스크립트 IDE 툴은 노드를 JS 실행도구로 사용하므로 불편하더라도 import 문법 사용 시 require 문법으로 바꾸는 작업이 선행되어야 한다. 이에 따라 발생하는 불편함은 환경설정을 통해 자동으로 트랜스파일링 작업이 수행되게 IDE 툴을 셋팅하면 최소화 할 수 있다.

# 1. CommonJS 방식

노드기반 서버사이드에서 주로 사용하는 모듈시스템이다.

## hello.js

```
var asia = function () {
    console.log('Hello Asia');
};

exports.korea = asia;

exports.world = function () {
    console.log('Hello World');
};
```

exports 키워드는 노드가 지원(require.js)하는 빌트인 객체다.

## main.js

```
var hello = require('./hello');

hello.world();
hello.korea();
```

노드 내장 모듈과 npm install 명령으로 설치해서 node\_modules 폴더에 위치한 모듈은 ./ 기호를 사용하지 않고 임포트한다.

"./" 기호는 현재 파일과 같은 위치에 있다는 의미이며 개발자가 추가로 만든 hello.js 파일을 임포트할 때 붙여야 한다. 파일명의 확장자 .js는 붙이지 않는다.

main.js 파일을 실행하여 정상적으로 hello.js에서 제공하는(exports) 자원을 사용할 수 있는지 확인한다.

다른 예제를 한번 더 살펴보자.

## hello2.js

```
module.exports = {  
  message: 'hello javascript!',  
  writer: 'Chris'  
};
```

외부에 제공할 자원이 하나라면 module.exports명령을 사용한다.

## main2.js

```
var {message, writer} = require('./hello2');  
  
console.log('message = '+message);  
console.log('writer = '+writer);
```

"require('./hello2')" 코드는 객체를 가리키므로 객체의 프로퍼티명을 사용하여 해체할당 할 수 있는 편리함이 있다.

## 2. ES6(ECMA2015) 방식

JS 표준방식이라는 의미를 갖는다. 기본적으로 노드의 V8엔진을 통해 JS 파일을 컴파일하기 때문에 웹스톰이나 아톰 같은 IDE 툴에서 노드가 지원하지 않는 ES6문법을 사용할 수는 없다.

물론, 컴파일러 엔진이 지원하는 문법이 점진적으로 늘어나고 있기 때문에 최신 문법을 그대로 사용할 수 있는지 없는지는 최신 기준으로 확인할 필요가 있다.

트랜스파일링을 통해 생성된 JS파일을 실행하는 방식으로 코드를 테스트 한다.

모듈을 정의하는 `export` 문법은 아래와 같다.

```
export name1, name2, ..., nameN;  
export *;  
export default name;
```

- name: export할 속성이나 함수, 객체
- \*: 파일 내의 정의된 모든 속성, 함수, 객체
- default: 모듈의 기본값. 파일 내에서 한 번만 호출될 수 있다.

속성을 export 하는 것은 CommonJS의 패턴과 동일한다.

```
export function foo() {};  
  
export {  
    foo: function () {},  
    bar: 'bar'  
};
```

ES6에 추가된 진보된 객체 리터럴(Enhanced Object Literals)을 함께 사용하면,

더 간결하게 변수를 export 할 수 있다.

```
var foo = function () {};  
var bar = function () {};
```

```
export { foo, bar };
```

모듈 전체를 export 하고자 할 땐, `default` 키워드를 함께 쓸 수 있다.

```
export default function () {}
```

`export default`는 CommonJS의 아래 패턴과 동일하다고 보면 된다.

```
module.exports = function () {};
```

export 한 모듈을 불러올 땐, `import` 키워드를 쓸 수 있고 문법은 아래와 같다.

```
import name from "module-name";
import { member [as alias], ... } from "module-name";
import "module-name" [as name];
```

- module-name: 불러올 모듈명, 파일명

- member: 모듈에서 export 한 멤버의 이름

- alias: 불러온 멤버를 할당할 변수명

`my-module` 모듈을 가져와 `myModule`라는 변수에 할당한다.

```
import myModule from 'my-module';
```

위 코드는 아래처럼 쓸 수도 있다.

```
import 'my-module' as myModule;
```

CommonJS 스타일로 작성했다면, 아래 코드와 동일하다.

```
var myModule = require('my-module');
```

모듈에서 특정 멤버만 변수로 할당할 수 있다.

```
import { foo, bar } from 'my-module';
```

CommonJS 스타일로 작성했다면, 아래 코드와 동일하다.

```
var myModule = require('my-module');
var foo = myModule.foo;
var bar = myModule.bar;
```

모듈이나 멤버의 이름을 다른 변수명으로 설정할 수 있다. 변수명이 긴 경우 요긴하게 사용할 수 있다.

```
import { reallyReallyLongModuleName as foo } from 'my-module';
```

단순히 특정 모듈을 불러와 실행만 할 목적이라면 아래처럼 `import`만 하는 것도 좋다.

주로 폴리필 라이브러리의 기능을 선택적으로 폴리필하고자 할 때 사용한다.

```
import 'my-module';
```

## Babel CLI 설치

```
npm install -g babel-cli
npm init -y
npm install --save-dev babel-latest
```

## 수동으로 Babel 트랜스파일링

### hello.js

```
var asia = function () {
    console.log('Hello Asia');
};

export let korea = asia;

export let world = function () {
    console.log('Hello World');
};
```

### main.js

```
import {world, korea} from './hello';

world();
korea();
```

## Babel 트랜스파일링

```
babel * -d build --ignore build --presets latest
```

\* : 현재 폴더 위치에 모든 파일을 대상으로 한다.

-d build : 작업결과를 build 폴더 밑에 생성한다.(-out-dir build)

--ignore build : 반복적인 트랜스파일링 작업 시 build 폴더를 무시하기 위한 옵션을 사용한다.

## build/hello.js

---

```
'use strict';

Object.defineProperty(exports, "__esModule", {
  value: true
});
var asia = function asia() {
  console.log('Hello Asia');
};

var korea = exports.korea = asia;

var world = exports.world = function world() {
  console.log('Hello World');
};
```

---

## build/main.js

---

```
'use strict';

var _hello = require('./hello');

(0, _hello.world)();
(0, _hello.korea);
```

---

위 JS 파일을 실행하는 것으로 코드의 정상작동을 확인한다.

트랜스파일링 작업에 익숙해지기 위해서 한번 더 연습해 보자.

### my-module.js

```
export default function cube(x) {
    return x * x * x;
}
```

### my-main.js

```
import cube from './my-module';
console.log(cube(3)); // 27
```

## Babel 트랜스파일링

```
babel my-* -d build --ignore build
```

### build/my-module.js

```
"use strict";

Object.defineProperty(exports, "__esModule", {
    value: true
});
exports.default = cube;
function cube(x) {
    return x * x * x;
}
```

### build/my-main.js

```
'use strict';

var _myModule = require('./my-module');

var _myModule2 = _interopRequireDefault(_myModule);

function _interopRequireDefault(obj) { return obj && obj.__esModule ? obj : { default: obj }; }

console.log((0, _myModule2.default)(3)); // 27
```

위 JS 파일을 실행하는 것으로 코드의 정상작동을 확인한다.

# 자동으로 Babel 트랜스파일링 : ATOM 사용

## 1. Babel의 필요성 확인

### hello.js

```
var asia = function () {
    console.log('Hello Asia');
};

export let korea = asia;

export let world = function () {
    console.log('Hello World');
};
```

### hello-importer.js

```
import {world, korea} from './hello';

world();
korea();
```

### 테스트 실행

위 파일에서 실행(alt+r)하면 에러가 발생한다.

```
D:\sooup\atom\workspace\hello-es6\es-import\hello.js:7
export let korea = asia;
^ ^ ^ ^ ^
SyntaxError: Unexpected token export
... 생략
```

노드가 지원하지 않는 export/import 문법을 사용할 수 없음을 파악했다. 바벨을 사용해서 6to5 작업을 통해 실행할 수 있는 소스코드(노드가 지원하는 문법의 코드)를 얻도록 설정해 보자.

참고로 언급하자면 노드 버전 4.0부터 ES6 문법인 class 키워드를 지원한다. 따라서 ATOM에서 js 확장자로 파일을 만들고 그 안에서 class 문법을 사용하고 바로 실행하는 것이 가능하다.

## 2. language-babel 패키지

### 2.1. 설치

```
File > Settings > Install >  
type 'language-babel' in search box and enter > Install
```

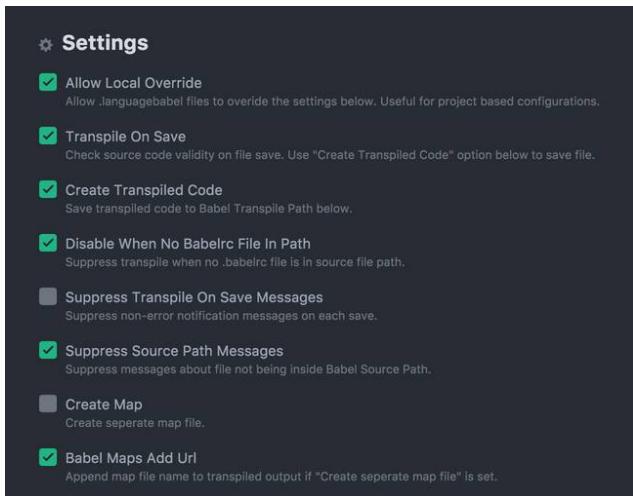
### 2.2. 패키지 기능 설명

<https://atom.io/packages/language-babel>

### 2.3. 패키지 설정

```
File > Settings > Packages >  
type 'language-babel' in Installed Packages Search Box >  
Community Packages > click 'Settings' button
```

다음화면을 참고하여 설정을 변경한다.



### 3 프로젝트 환경 설정

#### 3.1. package.json 파일 생성

```
npm init --yes
```

#### 3.2. 바벨 디펜던시 설치

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

#### 3.3. 바벨 설정파일 작성

##### .babelrc

```
{
  "only": [
    "*.babel",
    "*.jsx",
    "*.es6"
  ],
  "presets": [
    "es2015"
  ]
}
```

```
}
```

### .languagebabel

```
{
  "babelMapsPath": "",
  "babelMapsAddUrl": false,
  "babelSourcePath": "",
  "babelTranspilePath": "",
  "createMap": false,
  "createTargetDirectories": false,
  "createTranspiledCode": true,
  "disableWhenNoBabelrcFileInPath": true,
  "suppressSourcePathMessages": false,
  "suppressTranspileOnSaveMessages": false,
  "transpileOnSave": true
}
```

For most projects, it is better to configure language-babel via project-based **.languagebabel** file properties which will override the package settings.

### Transpile On Save

On any file save of a language-babel grammar enabled file the Babel transpiler is called. No actual transpiled file is saved but any Babel errors and/or successful indicators are notified by an ATOM pop-up. Not all files are candidates for transpilation as other settings can affect this. For example see Disable When No Babelrc File In Path and Babel Source Path below.

```
{"transpileOnSave": true} or {"transpileOnSave": false}
```

.languagebabel 파일의 기타 설정은 다음 사이트를 참고한다.

<https://atom.io/packages/language-babel>

### **3.4. 트랜스파일링 작업대상으로 설정하기**

hello.js/ hello-importer.js 파일의 확장자를 .es6로 모두 바꾼다.

파일을 저장하는 단축키(ctrl-s)를 누르면 바벨이 트랜스파일링을 진행하여 파일명은 같고 확장자가 js인 파일을 같은 폴더위치에 생성한다.

### **3.5. 테스트**

확장자가 js인 파일을 실행(alt+r)한다.

## **4. 작업결과 화면**

The screenshot shows the Atom code editor interface. On the left is a file tree with a dark theme. It contains a 'lesson' folder which includes an 'ecma6script' folder, an 'ecma6' folder, and a 'node\_modules' folder. Inside 'ecma6script', there's a 'module-system' folder containing 'ecma6' and 'main.js'. Inside 'ecma6', there are four files: 'hello.es6', 'hello.js', 'main.es6', and 'main.js'. The 'main.js' file is currently open in the main editor area. The code in 'main.js' is:

```
'use strict';
var _hello = require('./hello');
(_hello.world)();
(_hello.korea)();
```

To the right of the editor is a terminal window titled 'Atom Runner: main.js'. It displays the output of the Node.js execution:

```
Hello World
Hello Asia
```

Below the terminal, it says:

```
Running: node
(cwd=D:\sooup\atom\lesson\ecma6script\module-system\ecma6\main.js pid=15592). Exited with code=0 in
0.182 seconds.
```

At the bottom of the interface, there are status indicators: 'File 0 Project 0 ✓ No Issues ecma6script\module-system\ecma6\main.js 7:1', 'LF 2 deprecations UTF-8 Babel ES6 JavaScript', and a small warning icon.

### 3. IIFE 방식

Immediately Invoked Function Expression은 즉시 실행 함수 표현식이라고 부른다.

IIFE 방식을 사용하면 완벽히 분리된 스코프를 갖는다.

소스코드를 복사해서 JS파일 어느 위치에 끼어 넣어 사용하거나 HTML을 통한 JS파일들 결합시 스코프 충돌에 대한 걱정없이 사용할 수 있다.

## module-iife.js

```
/*
자바스크립트 모듈
객체, 함수, 기타 컴포넌트의 묶음으로써 분리된 스코프를 갖는다.
*/
(function (global) {
    var add = function (x, y) {
        return x + y;
    };

    var sub = function (x, y) {
        return x - y;
    };

    var calc = {
        add: function (a, b) {
            return add(a, b);
        },
        sub: function (a, b) {
            return sub(a, b);
        }
    };

    global.calc = calc;
})(global);

console.log(calc.add(1, 2)); // 3
console.log(calc.sub(1, 2)); // -1
```

## 4. AMD

Asynchronous Module Definition의 약자다. AMD는 브라우저에서 모듈을 구현하기 위해 만든 명세로 웹 페이지 로딩을 차단하지 않고도 비동기적으로 모듈을 임포트할 수 있다. 브라우저에 내장된 기능이 아니므로 AMD 라이브러리가 필요하고 require.js는 가장 잘 알려진 AMD 라이브러리이며 모듈은 모두 별도 파일로 구현해야 한다.

## 동적 로딩

참고: <http://d2.naver.com/helloworld/591319>

<script> 태그는 페이지 렌더링을 방해한다. <script> 태그의 HTTP 요청과 다운로드, 파싱(Parsing), 실행이 일어나는 동안 브라우저는 다른 동작을 하지 않는다. 브라우저 입장에서는 당연하고 안전한 동작 방식이지만 사용자 입장에서는 빨리 화면이 보이고 버튼이 동작하기를 바랄 뿐이다. 그래서 최적화 기법 중의 하나로 <script> 태그를 가능한 한 <body> 태그의 마지막에 배치하는 방법이 있다.

하지만 사용자의 첫 인터랙션이 가능할 때까지 걸리는 시간이 줄어들지는 않는다. 페이지를 더 빨리 렌더링할 수는 있어도 첫 렌더링과 첫 인터랙션에 필요하지 않은, 페이지에 필요한 모든 JavaScript를 로딩하기 때문이다. 화면이 복잡하고 AJAX로 점철된 웹 앱 수준의 규모에서는 이 시간이 큰 폭으로 커진다. 웹 앱은 AJAX로 전환되는 여러 뷰(View)를 가지고 있는 경우가 흔하다. 더 최적화를 하자면 첫 렌더링과 인터랙션에 필요한 JavaScript만 먼저 로딩하고 후에 사용자의 반응에 따라 나머지를 로딩하는 점진적인 방식이 필요하다.

동적 로딩(Dynamic Loading, Lazy Loading이라고도 부른다)은 페이지 렌더링을 방해하지 않으면서 필요한 파일만 로딩할 수 있다. 이를 구현하는 방법 중 하나로 <script> 태그의 동적 삽입이 있다. 이는 JavaScript로 <script> 태그를 생성하여 추가하는 방법이다. 이 외에도 XMLHttpRequest, document.write(), defer 같은 방법이 있지만 범용적으로 사용하기에는 치명적인 단점이 하나씩은 있어서 <script> 태그의 동적 삽입이 제일 안전하고 합리적이다.

### math.js

```
define("calc", [], function () {
    var sum = function (x, y) {
        return x + y;
    };

    var sub = function (x, y) {
        return x - y;
    };

    return {
        getSum : function (a, b) {
            return sum(a, b);
        },
    };
});
```

```
getSub : function (a, b) {  
    return sub(a, b);  
}  
};  
});
```

---

## index.js

```
// 디펜던시 calc가 먼저 로드된 뒤에 콜백이 수행된다.  
requirejs(['calc'], function (calc) {  
    console.log(calc.getSum(1, 2)); // 3  
    console.log(calc.getSub(1, 2)); // -1  
});
```

---

## index.html

```
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="utf-8">  
    <title>AMD</title>  
</head>  
<body>  
    <h1>AMD Running Test</h1>  
    <h3>check out console</h3>  
    <script src="https://cdnjs.cloudflare.com/ajax/libs/require.js/2.3.2/require.min.js"></script>  
    <script src="math.js"></script>  
    <script src="index.js"></script>  
</body>  
</html>
```

---

## 10. 함수 축약 표현식

### function-sugar-code.js

```
var obj = {
  render() {
    console.log('hello world');
  }
}

obj.render();
```

### 바벨 트랜스파일링

```
babel --presets latest function-sugar-code.js -d build
```

### 결과 확인

```
'use strict';

var obj = {
  render: function render() {
    console.log('hello world');
  }
}

obj.render();
```

"render()" 은 "render: function render()" 코드를 줄여 쓰는 문법이다.

# 11. Arrow function

애로우함수는 선언하는 문법만 다른 것이 아니라 많은 부분에서 일반적인 함수와 다르므로 명확한 이해가 필요하다.

- 애로우 함수는 선언된 위치를 기준으로 가장 가까운 스코프의 소유주(Context)에 자동으로 바인딩된다. 바인딩은 고정되며 바뀌지 않는다.
- 애로운 함수는 이름을 가질 수 없으며 언제나 익명함수로 선언해서 사용한다.
- 애로운 함수는 생성자로 사용될 수 없다. 따라서 new 키워드와 같이 사용할 수 없다.
- 애로운 함수는 prototype 프로퍼티가 없다. 따라서 상속관계를 맺을 수 없다.
- 애로운 함수는 arguments 객체를 갖고 있지 않다. 따라서 파라미터를 명시적으로 선언해야만 전달되는 파라미터를 받아서 사용할 수 있다.

## fat-arrow-function.js

```
var foo1 = () => {return 1};      // without any parameters
var foo2 = a => {return a*2};     // with single parameter
var foo3 = (a,b) => {return a+b}; // with multiple parameter
console.log(foo1()); // 1
console.log(foo2(2)); // 4
console.log(foo3(3, 4)); // 7
```

## Return in arrow function

애로우 함수는 바디부분의 범위연산자 "{}"를 명시적으로 사용하지 않으면 return 구문이 선언된 것처럼 행동한다. 사실 자바스크립트는 함수 끝에서 언제나 return 하는 행동방식을 보인다. 다만 일반 함수에서 return 구문을 생략하면 "return;" 과 같이 행동하지만 애로우 함수에서는 "return 처리결과" 와 같이 행동한다는 차이가 있을 뿐이다.

명시적으로 함수의 범위연산자 "{}"를 사용하면 컴파일러는 개입하지 않는다.

### fat-arrow-function2.js

```
// because there is no block statement, hence JS implicit `return`  
var fn = a => a*2;  
console.log(fn(1)); // 2  
  
// because there is a block statement, and JS depends on explicit `return`;  
var fn = a => {a*2};  
console.log(fn(1)); // undefined  
  
// because there is a block statement, and we are explicitly returning  
var fn = a => {return a*2};  
console.log(fn(1)); // 2  
  
// because there is no curly braces,  
// but parantheses, which is considered as expression,  
// hence it implicitly `return` the object  
var fn = a => ((id:a));  
console.log(fn(1)); // {id:1}
```

## How this is different for arrow functions

애로우 함수는 일반함수처럼 상황에 따라 별도의 컨텍스트 바인딩을 가질 수 없다. 애로우 함수에서 this는 애로우 함수 위치를 기준으로 애로우 함수를 소유하고 있는 소유주를 찾아서 소유주의 this를 자신의 this로 삼는다.

다음은 일반 함수안에서 this의 행동방식을 살펴보는 예제이다.

```
global.a = 'global';

function foo() {
    console.log(this.a);
}

foo.a = 'func';

foo(); // global
foo.call(foo); // func
```

다음 예제는 잘 못 이해하기 쉽다.

```
// The callback function is called from the setTimeout function,
// and the setTimeout function is defined in the global scope,
// which means, that the callback function is called from the global scope.
function foo() {
    setTimeout(function(){
        console.log(this.a); // undefined
    },100);
}

foo.a = 'func';

foo.call(foo);
```

setTimeout 비동기 함수가 사용하는 콜백함수는 함수코드만이 큐에 저장되었다가 0.1초 후에 기동한다. 이 때에 콜백 함수는 호출자가 없이 함수만을 실행하는 것이 되므로 this가 가리키는 대상은 call 함수로 전달한 객체 foo가 아니다. 결과적으로 "undefined"가 출력된다.

bind 메소드를 사용하여 함수가 참조할 this를 미리 고정시킬 수 있다.

```
function foo() {
    setTimeout(function(){
        console.log(this.a); // func
    }).bind(this),100);
}

foo.a = 'func';
foo.call(foo);
```

일반함수를 bind 메소드로 호출자 this를 고정시키는 방법은 앞서서 살펴보았다. 애로우 함수를 사용하면 보다 간단히 같은 결과를 얻을 수 있다. 왜냐하면 애로우 함수의 위치에서 가장 까운 스코프의 소유주는 foo 함수이기 때문에 애로우 함수에서의 this는 foo 함수가 호출될 때 참조하는 호출자 this로 자동적으로 바인딩되기 때문이다.

```
function foo() {
    setTimeout(()=>{
        console.log(this.a); // func
    },100);
}

foo.a = 'func';
foo.call(foo);
```

"foo.call(foo);"라는 코드를 살펴보면 함수의 호출자는 foo 객체다. 따라서 애로운 함수의 this는 자동적으로 호출자 foo 객체를 가리킨다.

이번에는 반대로 애로우 함수로부터 출발해서 설명하겠다. 애로우 함수의 this를 처리하기 위해서 컴파일러는 애로우 함수 스코프를 살핀다. 애로우 함수는 단독으로 자신만의 바인딩을 별도로 가질 수 없으므로 찾을 수 없다. 다음으로 애로우 함수를 감싸고 있는 외부 스코프를 뒤진다. foo 함수는 일반함수로써 this를 가지고 있다. 이제 this를 찾았으므로 애로우 함수의 this를 foo 함수의 this를 가리키는 것으로 바인딩한다.

이해를 돋기 위해 위 예제를 살짝 변경한 아래 코드와 비교하면서 살펴보자.

```
function foo() {
    var that = this;
    setTimeout(()=>{
        console.log(that.a); // func
    },100);
}
```

```
    },100);
}

foo.a = 'func';
foo.call(foo);
```

---

이번에는 애로우 함수를 멤버로 갖고 있는 객체에 대한 예제를 살펴보자.

### arrow-as-object-member.js

---

```
global.a = 'global';

var obj = {
  a: 'object',
  foo: () => {
    console.log(this.a)
  },
  print: function(){
    console.log(this.a);
  }
};

obj.foo(); // undefined
obj.print(); // object

console.log('-----');

var foo = () => {
  console.log(this.a)
};

var print = function(){
  console.log(this.a);
```

```
}
```

---

```
foo(); // undefined
print(); // global
```

테스트 결과를 살펴보면 애로우 함수의 this는 글로벌 스코프와는 연관되지 않는다는 것을 알 수 있다.

애로우 함수의 this는 애로우 함수를 갖고 있는 큰 함수의 this와만 바인딩된다는 것을 알 수 있다.

따라서 위 예제에서 애로운 함수가 제대로 작동하게 하려면 함수의 스코프가 필요하므로 다음과 같이 변경하면 제대로 작동하는 코드를 얻을 수 있다.

---

```
var obj = {
  a: 'object',
  foo: function () {
    return () => {
      console.log(this.a)
    }
  }
};

obj.foo(); // object
```

애로우 함수는 arguments 객체를 지원하지 않는다.

---

```
function foo(a, b, c) {
  return (x, y) => {
    console.log(arguments); // { '0': 1, '1': 2, '2': 3 }
  }
}

foo(1, 2, 3)(4, 5);
```

애로우 함수에서는 파라미터 자리에 변수를 명시해야만 함수 호출 시 전달하는 파라미터를 이용할 수 있다. 따라서 arguments 객체가 없는 애로우 함수에서는 나머지 연산자를 사용하는 것이 편리하다.

---

```
function foo(a, b, c, ...restOfFoos) {
  console.log(restOfFoos); // [0]

  return (x, y, ...restOfArrows) => {
    console.log(restOfArrows); // [6,7]
  }
}

foo(1, 2, 3, 0)(4, 5, 6, 7);
```

---

## use case

애로운 함수는 콜백함수가 사용되는 모든 부분에서 맹활약을 펼칠 수 있다.

---

```
var smartPhones = [
  {name: 'iphone', price: 649},
  {name: 'Galaxy S6', price: 576},
  {name: 'Galaxy Note 5', price: 489}
];

// ES5
console.log(smartPhones.map(
  function (smartPhone) {
    return smartPhone.price;
  }
)); // [649, 576, 489]

// ES6
console.log(smartPhones.map(
  smartPhone => smartPhone.price
)); // [649, 576, 489]
```

---

---

```
var numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

console.log(numbers.filter(function (number) {
  return number % 3 === 0;
}));
// [3, 6, 9]

console.log(numbers.filter(number => number % 3 === 0));
// [3, 6, 9]
```

---

## It Doesn't Work The Way You Think It Does

### it-does-not-work-the-way-u-think.js

```
this.test = "attached to the module";

var foo = {
    test: "attached to an object"
};

// method 프로퍼티는 메소드이름, 콜백함수를 받아서
// this가 가리키는 객체에 프로퍼티로 추가한다.
foo.method = function (name, cb) {
    this[name] = cb;
}

foo.method("bar", () => {
    console.log(this.test);
});

// 여러분이 기대한 대로 출력되는가?
foo.bar(); // attached to the module
```

"attached to an object"가 출력되도록 코드를 바꾸어 보자.

---

```
this.test = 'attached to the module';

var foo = {
    test: 'attached to an object'
};

foo.method = function (name, cb) {
    this[name] = cb;
};

foo.method('bar', function () {
    console.log(this.test);
});

foo.bar(); // attached to an object
```

---

애로우 함수가 모든 곳에 적합한 것은 아니라는 사실을 알 수 있다. 간단히, 콜백함수가 필요한 곳에서 사용한다고 생각하자. 객체의 프로퍼티로 메소드를 가리키도록 할 때 사용하는 것은 대부분 적합하지 않다.

## 12. \$ expression

### \$-expression.js

```
var keyWord = 'react';

// use back-tick `
var url = `https://www.google.com/#newwindow=1&q=${keyWord}`;

console.log(url);
// url = https://www.google.com/#newwindow=1&q=react
```

```
babel --presets latest $-expression.js -d build
```

```
'use strict';

var keyWord = 'react';

var url = 'https://www.google.com/#newwindow=1&q=' + keyWord;

console.log(url = ' + url);
```

백틱으로 선언한 문자열은 들여쓰기, 탭, 줄바꿈이 유지된다.

```
var html = `

<div>
  <label>id</label>
</div>
`;

console.log(html);

// <div>
//   <label>id</label>
// </div>
```

# 13. class

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>

## 클래스 주요 특징

1. class declarations are **not hoisted**.

클래스 문법으로 선언한 함수는 호이스팅되지 않는다. 따라서, 밑에서 선언한 클래스를 위에서 호출하는 만행을 저지할 수 없다.

2. brackets {}.

This is where you define class members, such as methods or constructors.

The bodies of class declarations and class expressions are **executed in strict mode**.

클래스 문법은 클래스 범위연산자안에서 새 객체의 멤버 프로퍼티, 함수가 제공하는 상속자원, 함수의 정적자원을 모두 모아서 선언하는 것이다.

클래스 범위안에 선언한 자원은 엄격모드에서 실행된다.

3. Constructor

The constructor method is a special method for creating and initializing an object created with a class.

A constructor can **use the super keyword** to call the constructor of a parent class.

클래스 문법에서는 새 객체의 멤버 프로퍼티를 정의하기 위한 생성자함수 constructor를 사용한다.

## 클래스 기본 문법

1. constructor 함수안에서 this 키워드로 선언된 변수는 new 키워드로 만들어지는 새 객체의 프로퍼티가 된다.
2. 함수들은 생성자.prototype 객체에 메소드로 추가된다.

ES5 이전에는 생성자 함수를 선언한 후 생성자.prototype 객체에 메소드를 추가했었다.

추가로 getter, setter 접근자 메소드를 정의해서 사용할 수 있다.

3. 함수앞에 static 키워드를 붙이면 함수객체 자신의 프로퍼티로 추가된다.

## class1.js

```
class Polygon {
    constructor(height, width) {
        let writer = 'Chris';
        this.height = height;
        this.width = width;
    }

    // 게터, 세터 메소드는 함수처럼 작성하지만 변수처럼 사용한다.
    get area() {
        return this.calcArea();
    }

    calcArea() {
        return this.height * this.width;
    }

    static help() {
        console.log('new Polygon(height, width)');
    }
}

const p = new Polygon(10, 10);

/*
constructor 함수안에서 this로 정의한 자원은
new 키워드로 새로 만들어지는 객체에 프로퍼티로 추가된다.
*/
console.log(p); // {"height":10,"width":10}

console.log(p.area); // 100
// 게터, 세터 메소드는 ()를 사용하지 않고 변수처럼 사용한다.
console.log(p.calcArea()); // 100

/*
클래스안에 선언된 함수들은 클래스.prototype 객체에 메소드로 추가된다
*/
console.log(Object.getOwnPropertyNames(Polygon.prototype));
// [ 'constructor', 'area', 'calcArea' ]

console.log(typeof Polygon.prototype.area); // number
console.log(typeof Polygon.prototype.calcArea); // function

/*
함수앞에 static 키워드를 붙이면 함수객체 자신에 프로퍼티로 추가된다
*/
console.log(Object.getOwnPropertyNames(Polygon));
// [ 'length', 'name', 'prototype', 'help' ]
Polygon.help();
// new Polygon(height, width)
```

# 클래스 문법 코드를 ES5 코드로 트랜스파일링한 결과 확인

ES5로 트랜스파일링을 수행하기 위해서 콘솔에서 다음 명령을 실행한다.

```
babel --presets latest class-usecase.js -d build
```

## build/class1.js

```
/*
  클래스는 strict 모드에서 수행된다.
 */
'use strict';

var _typeof = typeof Symbol === "function" && typeof Symbol.iterator === "symbol" ?
  function (obj) {
    return typeof obj;
  } : function (obj) {
    return obj && typeof Symbol === "function" && obj.constructor === Symbol && obj !==
Symbol.prototype ?
      "symbol" : typeof obj;
};

var _createClass = function () {
  function defineProperties(target, props) {
    for (var i = 0; i < props.length; i++) {
      var descriptor = props[i];
      descriptor.enumerable = descriptor.enumerable || false;
      descriptor.configurable = true;
      if ("value" in descriptor) descriptor.writable = true;
      Object.defineProperty(target, descriptor.key, descriptor);
    }
  }
  return function (Constructor, protoProps, staticProps) {
    if (protoProps) defineProperties(Constructor.prototype, protoProps);
    if (staticProps) defineProperties(Constructor, staticProps);
    return Constructor;
  };
}();

function _classCallCheck(instance, Constructor) {
  if (!(instance instanceof Constructor)) {
    throw new TypeError("Cannot call a class as a function");
  }
}

var Polygon = function () {
  function Polygon(height, width) {
    _classCallCheck(this, Polygon);
  }
}
```

```

/*
constructor 함수선언 부분과의 비교
-----
constructor(height, width) {
    let writer = 'Chris';
    this.height = height;
    this.width = width;
}
*/
var writer = 'Chris';
this.height = height;
this.width = width;
}

// 게터, 세터 메소드는 함수처럼 작성하지만 변수처럼 사용한다.

/*
_createClass 함수 호출 : class 문법 처리
-----
파라미터 1: 클래스명
파라미터 2: 함수 배열(게터/세터 포함)
파라미터 3: 정적 메소드 배열
*/
_createClass(Polygon, [
    {
        key: 'calcArea',
        value: function calcArea() {
            return this.height * this.width;
        }
    },
    {
        key: 'area',
        get: function get() {
            return this.calcArea();
        }
    },
    [
        {
            key: 'help',
            value: function help() {
                console.log('new Polygon(height, width)');
            }
        })
    ]
]);

return Polygon;
}

var p = new Polygon(10, 10);

```

---

## extends 키워드

### class-extends.js

```
class Animal {
    constructor(name) {
        this.name = name;
    }

    speak() {
        console.log(this.name + ' makes a noise.');
    }
}

class Dog extends Animal {
    speak() {
        console.log(this.name + ' barks.');
    }
}

var d = new Dog('Jindo');
d.speak();
// Jindo barks.

console.log(d);
// {"name":"Jindo"}

console.log(d.__proto__ === Dog.prototype);
// true
console.log(Object.getOwnPropertyNames(Dog.prototype));
// [ 'constructor', 'speak' ]

console.log(Dog.prototype.__proto__ === Animal.prototype);
// true
console.log(Object.getOwnPropertyNames(Animal.prototype));
// [ 'constructor', 'speak' ]

/*
babel --presets latest class-extends.js -d build
*/
```

## ES5, ES6 문법 혼용

함수는 다른 함수 또는 객체를 상속받을 수 있다.

class 문법과 기존의 함수 상속 문법을 섞어서 사용할 수 있다.

### class-extends-es5.js

```
function Animal(name) {
  this.name = name;
}

Animal.prototype.speak = function () {
  console.log(this.name + ' makes a noise.');
};

var animal = new Animal('Animal');
animal.speak(); // Animal makes a noise.
animal.__proto__.speak.call(animal); // Animal makes a noise.

console.log('-----');

class Dog extends Animal {
  speak() {
    console.log(this.name + ' barks.');
  }
}

var dog = new Dog('Dog');
dog.speak(); // Dog barks.
dog.__proto__.speak.call(dog); // Dog barks.
dog.__proto__.__proto__.speak.call(dog); // Dog makes a noise.

console.log('-----');

function Pet(name) {
  this.name = name;
}

Pet.prototype.speak = function () {
  console.log(this.name + ' barks bowwow.');
};

Pet.prototype.__proto__ = Dog.prototype;

var pet = new Pet('Pet');

pet.speak(); // Pet barks bowwow.
pet.__proto__.speak.call(pet); // Pet barks bowwow..
pet.__proto__.__proto__.speak.call(pet); // Pet barks.
pet.__proto__.__proto__.__proto__.speak.call(pet); // Pet makes a noise.
```

## 클래스가 객체를 대상으로 상속

### class-inherit-regular-object.js

```
var log = console.log;

var Animal = {
  speak() {
    console.log(this.name + ' makes a noise.');
  }
};

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

  speak() {
    console.log(this.name + ' barks.');
  }
}

/*
클래스는 비 컨스트럭터블(non-constructible) 객체를 상속할 수 없다.
객체를 상속하기 위해서는 Object.setPrototypeOf() 메소드를 사용해야 한다.

Object.setPrototypeOf(자식 객체, 부모 객체)
*/
Object.setPrototypeOf(Dog.prototype, Animal);
log(Dog.prototype.__proto__ === Animal); // true

var d = new Dog('Jindo');
d.speak();

log(Object.getOwnPropertyNames(d));
// [ 'name' ]
log(Object.getOwnPropertyNames(d.__proto__));
// [ 'constructor', 'speak' ]
log(Object.getOwnPropertyNames(d.__proto__.__proto__));
// [ 'speak' ]
```

## super 키워드

### class-super.js

```
var log = console.log;

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

    speak() {
        console.log(this.name + ' makes a noise.');
    }
}

/*
super 키워드로 부모가 갖고 있는 함수를 호출할 수 있다.
*/
class Lion extends Cat {
    speak() {
        super.speak();
        console.log(this.name + ' roars.');
    }
}

var lion = new Lion('leo');

log(lion.name);
// leo
lion.speak();
// leo makes a noise.
// leo roars.
```

## class-super2.js

```
class Cat {
  constructor(state = {}, props = {}) {
    console.log('Cat > constructor() called.');
    this.state = state;
    this.props = props;
  }
}

class Lion extends Cat {
  /*
    클래스 내 constructor 함수를 명시하지 않으면
    super()를 호출하는 constructor 함수가 있는 것과 같다.
  */
  constructor(state, props, data = []) {
    /*
      super()의 기능은 Cat.call(this)와 같다.
    */
    super();
    /*
      문법적으로 super() 호출 없이 this를 사용할 수 없다.
      에러메시지 : 'this' is not allowed before super()
    */
    this.data = data;
    console.log('Lion > constructor() called.');
  }
}

var lion = new Lion({a: 1}, {b: 2}, [1, 2, 3]);
// Cat > constructor() called.
// Lion > constructor() called.

console.log(lion);
// { state: {}, props: {}, data: [ 1, 2, 3 ] }
```

# 클래스 표현식 vs 선언식

## class-super-super.js

```
let log = console.log;

/*
1. 클래스 표현식
class 키워드 다음 클래스명을 지정하지 않는다.
*/
let Korean = class {
    constructor(a) {
        // new 키워드로 새로 만들어지는 객체에 추가된다.
        this.a = a;
    }

    // 메소드는 prototype 객체에 추가된다.
    printA() {
        console.log('a = ' + this.a);
    }

    // 정적메소드는 함수객체에 추가된다.
    static say(caller) {
        console.log('Hello ' + caller.a);
    }
};

/*
2. 클래스 선언식
*/
class HongFamily extends Korean {
    constructor(a, b) {
        super(a);
        this.b = b;
    }

    printB() {
        console.log('b = ' + this.b);
    }

    fetchB() {
        return this.b;
    }

    // getter, setter 메소드는 일반적으로 메소드명을 중복해서 선언한다.
    get bValue() {
        return this.b;
    }

    set bValue(b) {
        this.b = b;
    }
}
```

```
        }
    }

class Gildong extends HongFamily {
    constructor(a, b, c) {
        super(a, b);
        this.c = c;
    }

    printC() {
        console.log('c = ' + this.c);
    }
}

let gildong = new Gildong(1, 2, 3);

log(gildong);
// { a: 1, b: 2, c: 3 }

log(Object.getOwnPropertyNames(Gildong.prototype));
// [ 'constructor', 'printC' ]

gildong.printC();
// c = 3

log(Object.getOwnPropertyNames(HongFamily.prototype));
// [ 'constructor', 'printB', 'fetchB', 'bValue' ]

gildong.printB();
// b = 2

log(Object.getOwnPropertyNames(Korean.prototype));
// [ 'constructor', 'printA' ]

gildong.printA();
// a = 1

log(-----);

Korean.say(gildong); // Hello 1

log(-----);

log(gildong.fetchB()); // 2

log(gildong.b); // 2

log(gildong.bValue); // 2
```

---

## ES5 상속과 class를 사용한 ES6 상속의 차이점

<https://www.sitepoint.com/object-oriented-javascript-deep-dive-es6-classes/>

JavaScript's class syntax is often said to be **syntactic sugar**, and in a lot of ways it is,

but there are also real **differences** things **we can do with ES6 classes** that **we couldn't with ES5**.

### Static Properties Are Inherited

---

```
// ES5
function B() {}
B.f = function () {};

function D() {}
D.prototype = Object.create(B.prototype);

D.f(); // error
```

---

---

```
// ES6
class B {
    static f() {}
}

class D extends B {}

D.f(); // ok
```

---

## Built-in Constructors Can Be Subclassed

```
// ES5
function D() {
    Array.apply(this, arguments);
}

D.prototype = Object.create(Array.prototype);

var d = new D();
d[0] = 42;

console.log(d instanceof Array); // true
console.log(d.length); // 0 - bad
```

```
// ES6
class D extends Array {}

let d = new D();
d[0] = 1;

console.log(d.length); // 1 - good
```

## Multiple Inheritance with Proxies

```
let transmitter = {
    transmit() {
        console.log('transmit() called');
    }
};

let receiver = {
    receive() {
        console.log('receive() called');
    }
};

// Create a proxy object that intercepts property accesses
// and forwards to each parent, returning the first defined value it finds
let inheritsFromMultiple = new Proxy([transmitter, receiver], {
    get: function(proxyTarget, propertyKey) {
        const foundParent = proxyTarget.find(parent => parent[propertyKey] !== undefined);
        return foundParent && foundParent[propertyKey];
    }
});

inheritsFromMultiple.transmit(); // transmit() called
inheritsFromMultiple.receive(); // receive() called

console.log(Array.isArray(inheritsFromMultiple)); // true
console.log(inheritsFromMultiple instanceof Array); // true
```

# 14. 비동기 처리

일반적으로 동기식으로 수행되는 언어는 멀티 스레드를 지원한다. 하지만 자바스크립트는 단일 스레드 언어다. 따라서 일부 로직이 무한루프에 빠진다면 전체 앱이 작동을 멈춘다.

비동기 연산의 결과를 노드앱에 전달하는 인기있는 방식은 다음과 같다.

- Call-back Function
- Event Emitter
- Promise

## 1. Call-back Function

다음은 fs 모듈을 사용하여 비동기적으로 파일 내용을 읽는다.

### readme.txt

```
i love u
```

### file-read.js

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

fs.readFile('readme.txt', 'utf8', function (error, data) {
  if (error) {
    return console.error(error);
  }
  console.log(data);
});
```

콜백함수는 비동기식 함수의 인자로 전달되며 비동기식 함수의 연산이 끝나면 연산 결과를 콜백함수의 파라미터로 받으면서 호출된다. 관례적으로 함수 파라미터 정의에서 콜백함수는 마지막 인자로 지정한다. **관례적으로 콜백함수의 파라미터 정의에서 error 정보를 담고 있는 인자는 첫 번째로 사용한다.**

이는 에러처리를 실수로 누락하는 일이 없도록 하기 위함이다.

이번에는 동기식 방식으로 파일 내용을 읽는 코드를 살펴보자.

### file-read-sync.js

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

try {
  var data = fs.readFileSync('readme.txt', 'utf8');
  console.log(data);
} catch (error) {
  console.error(error);
}
```

동기식 코드에서는 try 문을 사용하여 에러를 처리한다. 하지만, **비동기식 코드에서는 try 문으로 에러를 처리할 수 없다.** 이미 코드는 다음으로 진행되었고 결과는 나중에 리턴되기 때문이다.

그러므로 비동기식 코드에서 try 문은 존재 의미가 없다. 대안으로 예외처리 로직을 콜백함수에 포함시켜서 전달하고 비동기함수가 처리를 마치면 전달받은 콜백함수를 사용하는 방식으로 연동해서 처리한다.

## 콜백함수 사용 시 코드처리의 흐름

### guess-console-log.js

```
var fs = require("fs");
var sourceFile = './why-use-var-count.js';
var targetFile = './new-file-copied.js';

fs.chmod(sourceFile, 777, function (err) {
  if (err) {
    console.log(err.message);
    sourceFile = './new-file-copied.js';
    targetFile = './why-use-var-count.js';
  }

  console.log('1');

  fs.rename(sourceFile, targetFile, function (err) {
    console.log('2');

    fs.lstat(targetFile, function (err, stats) {
      console.log('3');
      console.log(stats.size);
    });

    console.log('4');

  });
  console.log('5');
});
```

콜백함수를 중첩해서 사용하면 코드 처리의 흐름을 놓치기 쉽다.

## 반복문에 의한 다수의 콜백함수들의 결과를 모아서 사용하기

### why-use-var-count.js

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

function calculateBytes() {
    fs.readdir(__dirname, function (err, filenames) {
        var count = filenames.length;
        var totalBytes = 0;
        var i;
        for (i = 0; i < filenames.length; i++) {
            fs.stat(__dirname + '/' + filenames[i], (function(fileName){
                return function (err, stats) {
                    console.log(fileName+' = '+stats.size);
                    totalBytes += stats.size;
                    count--;
                    if (count === 0) {
                        console.log('totalBytes = '+totalBytes);
                    }
                }
            })(filenames[i]));
        }
    });
}

calculateBytes();
```

count-- 같은 로직이 왜 필요한지 생각해 보자.

## 2. Event Emitter

비동기식 코드를 구현하는 두 번째 방법은 이벤트 전송자를 사용하는 것이다.

복수의 콜백함수 및 반복적으로 콜백함수를 사용해야 할 필요가 있을 때 적합한 방법이다.

emitter로 불리는 어떤 종류의 객체를 이벤트 이름으로 정의된 특정 이벤트에 정기적으로 전달해 listener로 불리는 함수 객체를 실행한다. 이벤트를 내보내는 모든 객체는 EventEmitter 클래스의 인스턴스다. 이 객체는 하나 이상의 함수를 이벤트로 사용할 수 있도록 이름을 넣어 추가하는 eventEmitter.on() 함수를 사용할 수 있다.

eventEmitter.on() 메소드로 등록된 리스너는 이벤트 이름이 호출되는 매 횟수만큼 실행된다.

eventEmitter.once() 메소드로 등록한 리스너는 호출한 직후 제거되어 다시 호출해도 실행되지 않는다.

### /event-emitter/counter.js

```
var EventEmitter = require('events').EventEmitter;

function Counter(limit) {
  var limit = limit || 1;

  EventEmitter.call(this);

  var self = this;
  var count = 0;

  this.start = function () {
    this.emit('start');

    var intervalId = setInterval(function () {
      ++count;
      self.emit('count', count);
      if (count >= limit) {
        clearInterval(intervalId);
      }
    }, 1000);
  }
}
```

```

};

}

Counter.prototype = Object.create(EventEmitter.prototype);
// 이용가능한 프로퍼티가 무엇이 있는지 확인해 보라.

exports.Counter = Counter;

```

## /event-emitter/use-counter.js

```

var Counter = require('./counter').Counter;

var stopWatch = new Counter(10);
// 객체의 프로퍼티가 무엇이 있는지 확인하고 왜 그렇게 구성되었는지 생각해 보자.

stopWatch.on('start', function () {
  console.log('started');
});

stopWatch.on('count', function (count) {
  console.log('count: ' + count);
});

stopWatch.start();

```

새 객체는 프로토타입체이닝 연결에 따라 EventEmitter의 프로토타입 객체에 있는 함수들을 사용할 수 있다. 위 경우는 on() 함수가 이 경우에 해당한다.

## /event-emitter/counter2.js

```
var util = require('util');
var EventEmitter = require('events').EventEmitter;

function Counter() {
  EventEmitter.call(this);

  var self = this;
  var count = 0;
  var intervalId = null;

  this.start = function (limit) {
    var limit = limit || 0;
    this.emit('start');

    intervalId = setInterval(function () {
      if (!limit) {
        self.emit('error', new Error('limit parameter missing.'));
        end();
        return;
      }

      ++count;
      self.emit('count', count);

      if (count >= limit) {
        end();
      }
    }, 1000);
  };

  this.stop = function () {
    end();
  };

  function end(){
    self.emit('stop');
    clearInterval(intervalId);
  }
}
```

```
}

util.inherits(Counter, EventEmitter);

exports.Counter = Counter;
```

## /event-emitter/use-counter2.js

```
var Counter = require('./counter2').Counter;

var stopWatch = new Counter();

stopWatch.on('start', function () {
    console.log('started');
});

stopWatch.on('count', function (count) {
    console.log('count: ' + count);
});

stopWatch.on('stop', function () {
    console.log('stoped');
});

stopWatch.on('error', function (error) {
    console.log(error.message);
});

stopWatch.start(4);
```

### 3. Promise

프라미스는 아직 통보받지 못한 값을 표현하는 객체다.

프라미스는 비동기식 함수를 나중에 호출하겠다는 약속을 담은 객체다.

프라미스 객체를 반환하여 처리의 흐름이 막힘없이 연속되게 만들고

미래에 콜백함수를 호출할 준비가 되었을 때

프라미스 객체에 설정된 콜백함수를 호출할 것임을 약속한다.

프라미스를 처음 생성하면 미결(pending) 또는 미이행(unfulfilled) 상태가 되며

관련된 비동기식 코드가 수행을 완료할 때까지 이 상태로 남는다.

비동기식 코드가 성공적으로 완료되면 프라미스는 이행(fulfilled) 상태로 변하며

비동기식 호출이 실패하면 거절(rejected) 상태로 변한다.

#### /promise/test1.js

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

var promise = new Promise(function(resolve, reject){
    fs.readFile('data.txt', 'utf-8', function(error, data){
        if (error) {
            return reject(error);
        }

        resolve(data);
    });
});

promise.then(function(result){
    console.log(result);
}, function(error){
    console.log(error.message);
});
```

## /promise/test2.js

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

var promise = new Promise(function(resolve, reject){
    fs.readFile('none.txt', 'utf-8', function(error, data){
        if (error) {
            return reject(error);
        }

        resolve(data);
    });
});

promise.then(function(result){
    console.log(result);
}).catch(function(error){
    console.log(error.message);
}).then(function(){
    console.log('The End');
});
```

## /promise/test3.js

```
new Promise(function (resolve, reject) {
    setTimeout(function () {
        resolve(10);
    }, 3000);
})
.then(function (num) {
    console.log('first then: ', num);
    return num * 2;
})
.then(function (num) {
    console.log('second then: ', num);
    return num * 2;
})
.then(function (num) {
```

```
    console.log('last then: ', num);
});
```

프라미스를 보다 쉽게 사용할 수 있는 모듈을 살펴보자. 다음 모듈을 먼저 설치하고 다음을 진행한다.

```
npm install --save bluebird
```

## /promise/test4.js

```
var Promise = require('bluebird');
var fs = Promise.promisifyAll(require('fs'));

fs.readFileAsync('data.txt')
  .then(function(fileData){
    return fs.writeFileAsync('message.txt', fileData);
  })
  .catch(function(error){
    console.log(error);
  })
  .finally(function(){
    console.log('done');
});
```

이제 동기함수가 프라미스 객체를 리턴하므로 코드를 보다 직관적으로 작성할 수 있다.

there are times when you trigger multiple async interactions  
but only want to respond when all of them are completed.  
that's where Promise.all comes in.

The Promise.all method takes an array of promises and **fires one callback once they are all resolved**.

## /promise/test5.js

```
var req1 = new Promise(function (resolve, reject) {
  setTimeout(function () {
    resolve('First!');
  }, 4000);
});

var req2 = new Promise(function (resolve, reject) {
  setTimeout(function () {
    reject('Second!');
  }, 3000);
});

Promise.all([req1, req2]).then(function (results) {
  console.log('Then: ', results);
}).catch(function (err) {
  console.log('Catch: ', err);
});
```

instead of waiting for all promises to be resolved or rejected,

Promise.race triggers as soon as any promise in the array is resolved or rejected.

## /promise/test6.js

```
var req1 = new Promise(function (resolve, reject) {
    setTimeout(function () {
        resolve('First!');
    }, 8000);
});

var req2 = new Promise(function (resolve, reject) {
    setTimeout(function () {
        resolve('Second!');
    }, 3000);
});

Promise.race([req1, req2]).then(function (one) {
    console.log('Then: ', one);
}).catch(function (one, two) {
    console.log('Catch: ', one);
});
```

## 15. TypeScript 소개

### Learn TypeScript in 30 Minutes

<http://tutorialzine.com/2016/07/learn-typescript-in-30-minutes/>

Today we're going to take a look at TypeScript, a **compile-to-JavaScript language** designed for developers who build large and complex apps. It **inherits many programming concepts from languages such as C# and Java** that add more discipline and order to the otherwise very relaxed and free-typed JavaScript.

This tutorial is aimed at people who are fairly proficient in JavaScript but are still beginners when it comes to TypeScript. We've covered most of the basics and key features while including lots of examples with commented code to help you see the language in action. Let's begin!

### The Benefits of Using TypeScript

JavaScript is pretty good as it is and you may wonder **Do I really need to learn TypeScript?**

Technically, you do not need to learn TypeScript to be a good developer, most people do just fine without it. However, working with TypeScript definitely has its benefits:

- Due to the **static typing**, code written in TypeScript is **more predictable**, and is generally **easier to debug**.
- Makes it easier to **organize the code** base for very large and complicated apps **thanks to modules, namespaces and strong OOP support**.
- TypeScript has a **compilation step to JavaScript** that **catches all kinds of errors before** they reach **runtime** and break something.
- The upcoming **Angular 2 framework is written in TypeScript** and it's recommended that developers use the language in their projects as well.



## Chapter 2 : Angular Basic

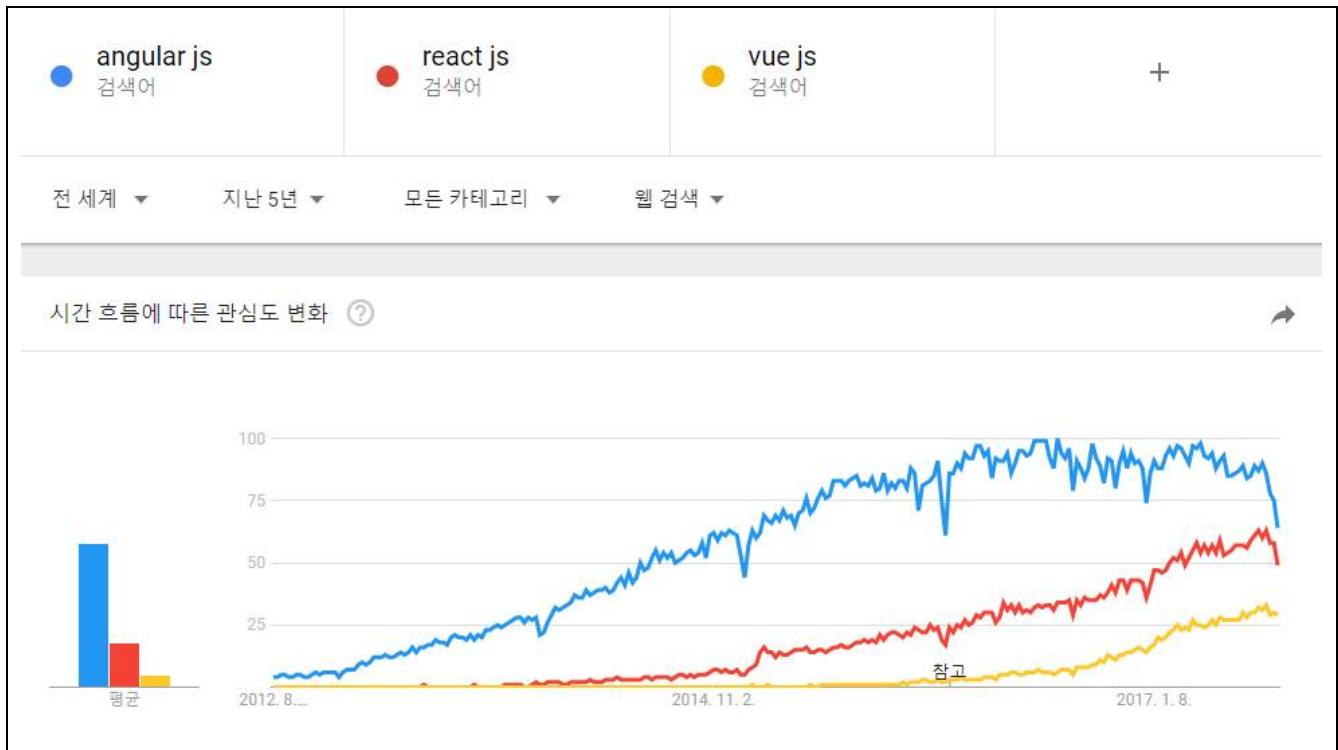
앵귤러 구조를 파악한 후 간단한 예제를 통해 빠르게 개념을 확립합니다.

앵귤러의 핵심 구성요소인 컴포넌트, 서비스, 파이프, 디렉티브, 모듈을 앵귤러 CLI 툴과 같이 학습합니다.

- 앵귤러 소개
- Simple Example
- Project Structure
- Angular CLI
- Component
- Service
- Pipe
- Directive
- Module

## Step 1 – 앵귤러 소개

앵귤러는 구글에서 만든 자바스크립트 프레임워크다. 2010년 10월 정식버전을 발표하였다. 2016년 9월 앵귤러2 버전을 발표하였다. 앵귤러2는 이전 버전의 처리속도를 개선하기 위해서 여러 단계를 거쳐 호출해야 하는 복잡한 단계를 제거했다. 더불어 자바스크립트의 장점인 높은 자율도에 의해서 발생되는 단점인 디버깅에 어려움을 해결하기 위해서 타입스크립트를 도입했다. 타입스크립트는 마이크로소프트가 2012년 10월 출시했다.



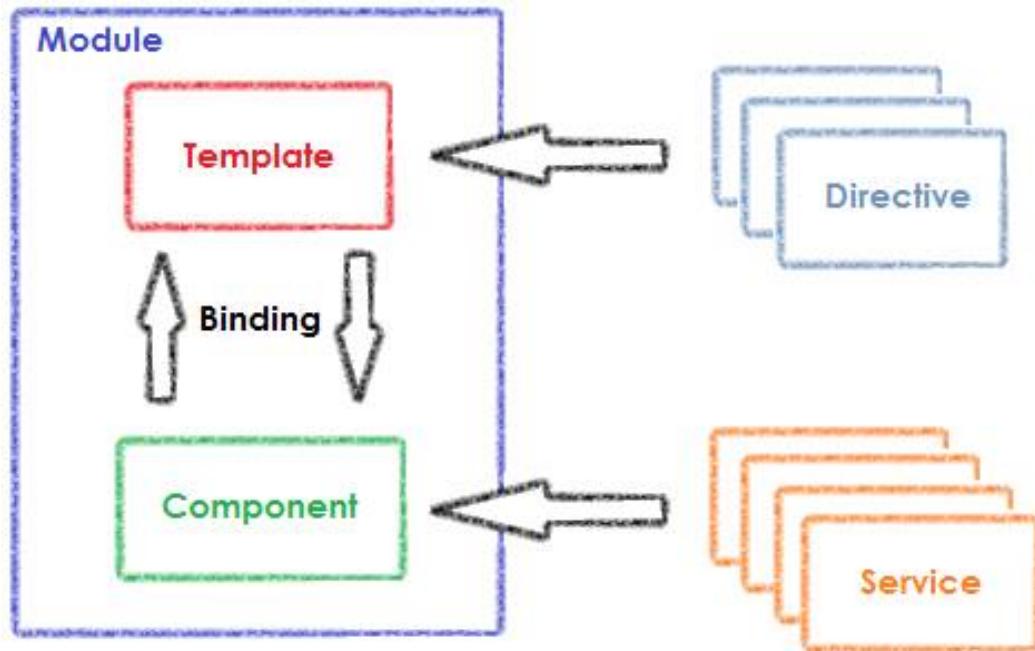
### 앵귤러를 사용해야 하는 이유

- 크로스 플랫폼 지원
- 처리속도
- 개발생산성

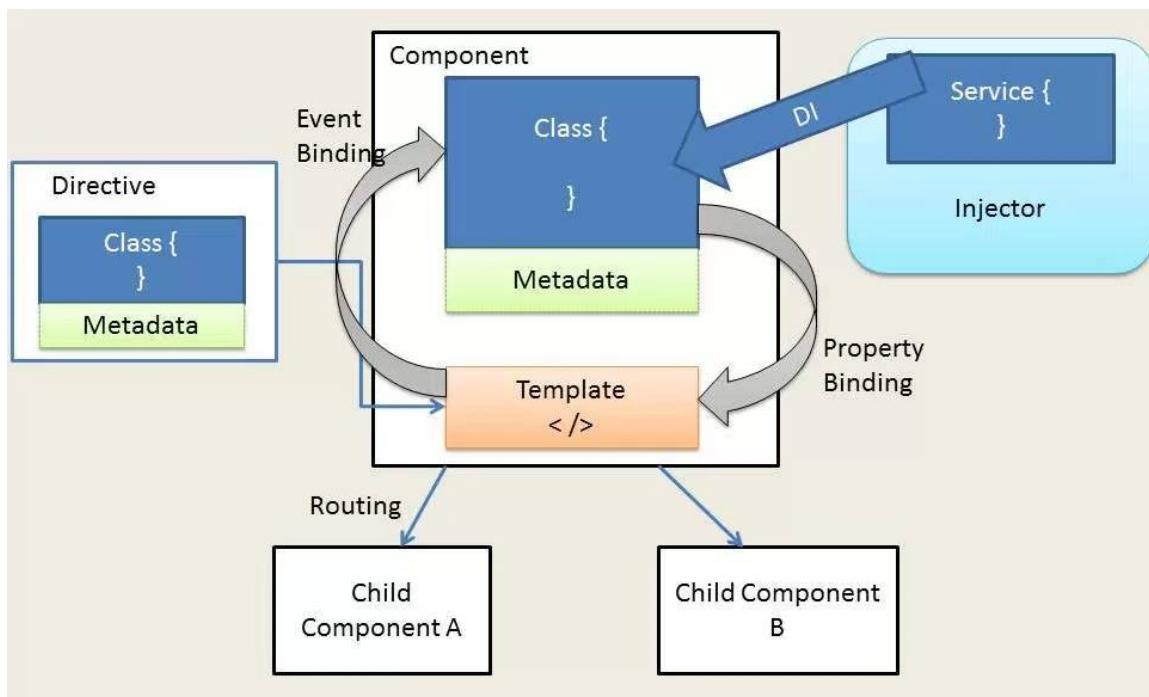
최근 들어 브라우저에서 동작하는 스크립트의 역할이 뷰에서 머물지 않고 컨트롤러까지 확대되었고 데이터를 자바스크립트 객체상태로 유지한다. 이에 따라 서버의 역할은 데이터를 제공하는 공급자 역할로 축소되고 있다.

## 앵글러 아키텍처

### ■ 간단한 아키텍처



### ■ 상세한 아키텍처



## ■ 앵귤러 핵심 구성요소

구성요소	책임
Module	화면의 구성요소를 묶어서 처리하는 앵귤러의 모듈시스템의 기본 단위
Component	템플릿과 바인딩으로 연결되며 필요한 로직을 처리하는 컨트롤러
Service	재사용하기 위해서 독립해서 정의하는 컴포넌트를 위한 처리 로직
Template	화면에 엘리먼트를 어떻게 배치할 것인지 정의하는 템플릿(엘리먼트들의 묶음)
Directive	재사용하기 위해서 독립해서 정의하는 엘리먼트를 위한 추가 기능

Angular는 HTML과 JavaScript 또는 JavaScript로 컴파일되는 TypeScript와 같은 언어로 클라이언트 응용 프로그램을 작성하기 위한 프레임 워크입니다.

프레임 워크는 몇 개의 라이브러리로 구성되며, 일부 라이브러리는 코어 라이브러리이고 일부 라이브러리는 선택적 라이브러리입니다.

Angularized 마크업을 사용하여 HTML 템플릿을 작성하고, 이 템플릿을 관리하기 위한 컴포넌트 클래스를 작성하고, 서비스에서 로직을 추가하고, 모듈로 컴포넌트 및 서비스를 묶어서 프로그램을 작성합니다.

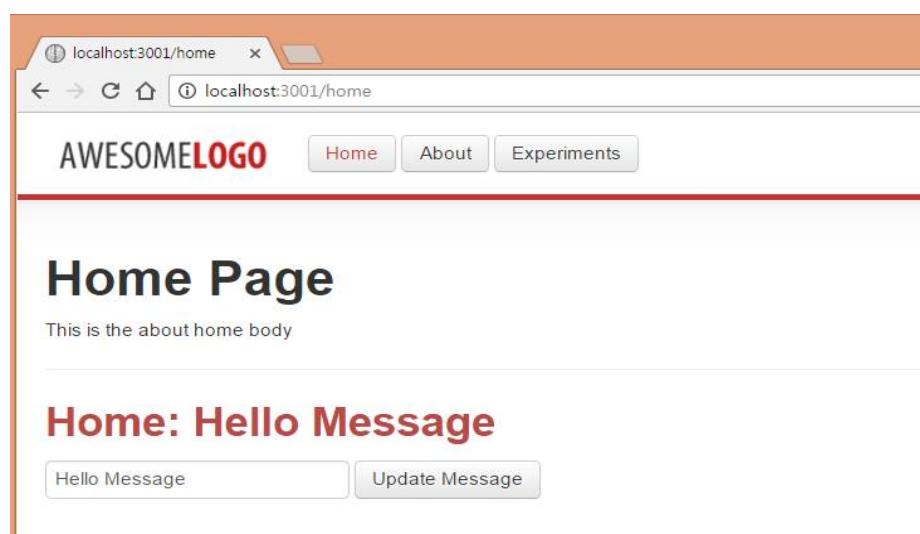
그런 다음 루트 모듈을 부트 스트랩하여 앱을 실행합니다. Angular는 브라우저에서 애플리케이션 콘텐츠를 제공하고 제공받은 지시에 따라 사용자와 상호 작용합니다.

## Step 2 – Simple Example

간단한 예제를 통해 바로 개념잡기

```
inspired by : https://github.com/simpulton/angular2-website-routes.git  
upgraded by : softcontext@gmail.com
```

```
git clone https://github.com/softcontext/angular-simple-demo  
cd angular-simple-demo  
npm install  
npm start
```



소스 코드를 통해 전체적인 프로젝트 구조 및 기동순서를 파악합니다.

# Step 3 – Project Structure

angular cli를 사용하면 환경설정과 관련한 중요한 파일은 다음 2개다.

## 디펜던시 관리

scripts 에 할당하는 객체는 자주 사용하는 명령어를 등록해 놓고 사용하는 기능이다.

### package.json

```
{
  "name": "angular-simple-demo",
  "version": "0.0.0",
  "license": "MIT",
  "scripts": {
    "ng": "ng",
    "start": "ng serve --open", // npm start
    "build": "ng build", // npm run build
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": { // 운영 시 필요한 디펜던시
    "@angular/animations": "^4.0.0",
    "@angular/common": "^4.0.0", // nglf, ngFor.. 구조 �렉티브, 서비스, 파이프, 디렉티브
    "@angular/compiler": "^4.0.0", // 앵귤러 컴파일 라이브러리
    "@angular/core": "^4.0.0", // 빌트인 애노테이션
    "@angular/forms": "^4.0.0", // 폼 처리 관련
    "@angular/http": "^4.0.0", // HTTP 처리 관련
    "@angular/platform-browser": "^4.0.0", // 브라우저 실행을 위해 공유 된 코드 (DOM 스레드, WebWorker)
    "@angular/platform-browser-dynamic": "^4.0.0",
    // 템플릿(바인딩, 컴포넌트 등) 및 리플렉티브 DI를 처리하는 클라이언트 측 코드
    "@angular/router": "^4.0.0", // 클라이언트 라우팅 처리
    "core-js": "^2.4.1", // 폴리필
    "rxjs": "^5.4.1", // 비동기 처리
    "zone.js": "^0.8.14" // 변경 감지(구글 제작)
  },
  "devDependencies": { // 개발 시 필요한 디펜던시
  }
}
```

```
"@angular/cli": "1.2.4",
"@angular/compiler-cli": "^4.0.0",
"@angular/language-service": "^4.0.0",
"@types/jasmine": "~2.5.53",
"@types/jasminewd2": "~2.0.2",
"@types/node": "~6.0.60",
"codemlyzer": "~3.0.1",
"jasmine-core": "~2.6.2", // 테스트 프레임워크
"jasmine-spec-reporter": "~4.1.0",
"karma": "~1.7.0", // 테스트 런너
"karma-chrome-launcher": "~2.1.1",
"karma-cli": "~1.0.1",
"karma-coverage-istanbul-reporter": "^1.2.1",
"karma-jasmine": "~1.1.0",
"karma-jasmine-html-reporter": "^0.2.2",
"protractor": "~5.1.2", // 통합테스트
"ts-node": "~3.0.4",
"tslint": "~5.3.2",
"typescript": "~2.3.3"
}
}
```

주의: .json 파일에 // 주석을 달 수 없다.

## 프로젝트 설정

기동파일 설정, 소스코드 위치 설정, 정적자원 설정, 테스트 설정 등을 관리하는 환경설정 파일이다.

### .angular-cli.json

```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "project": {
    "name": "angular-simple-demo"
  },
  "apps": [
    {
      "root": "src", // TS 파일 위치
      "outDir": "dist", // 빌드결과 저장 폴더
      "assets": [ // 정적자원 폴더
        "assets",
        "favicon.ico"
      ],
      "index": "index.html", // 기동 HTML 파일
      "main": "main.ts", // 기동 TS 파일
      "polyfills": "polyfills.ts", // 폴리필 설정
      "test": "test.ts",
      "tsconfig": "tsconfig.app.json",
      "testTsconfig": "tsconfig.spec.json",
      "prefix": "app", // 셀렉터 이름 접두어
      "styles": [ // 스타일 임포트
        "styles.css"
      ],
      "scripts": [], // 스크립트 임포트
      "environmentSource": "environments/environment.ts",
      "environments": {
        "dev": "environments/environment.ts",
        "prod": "environments/environment.prod.ts"
      }
    }
  ],
  "e2e": {
    "protractor": {
      "config": "./protractor.conf.js"
    }
  }
},
```

```
"lint": [
  {
    "project": "src/tsconfig.app.json",
    "exclude": "**/node_modules/**"
  },
  {
    "project": "src/tsconfig.spec.json",
    "exclude": "**/node_modules/**"
  },
  {
    "project": "e2e/tsconfig.e2e.json",
    "exclude": "**/node_modules/**"
  }
],
"test": {
  "karma": {
    "config": "./karma.conf.js"
  },
  "defaults": {
    "styleExt": "css",
    "component": {}
  }
}
```

## 처리 흐름

**npm start → package.json파일의 scripts에 선언된 스크립트 실행**

```
ng serve
```

### 개발 시 사용한 index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>AngularSimpleDemo</title>
    <base href="/">

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
  </head>
  <body>
    <app-root></app-root>
  </body>
</html>
```

앵귤러 CLI는 웹팩을 사용하여 동적으로 빌드한다. 그 결과, 다음과 같이 파일들이 웹팩 개발서버에 제공된다.

### 웹팩 개발서버에게 제공되는 index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>AngularSimpleDemo</title>
    <base href="/">

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
  </head>
  <body>
    <app-root></app-root>
```

```
<script type="text/javascript" src="inline.bundle.js"></script>
<script type="text/javascript" src="polyfills.bundle.js"></script>
<script type="text/javascript" src="styles.bundle.js"></script>
<script type="text/javascript" src="vendor.bundle.js"></script>
<script type="text/javascript" src="main.bundle.js"></script>
</body>
</html>
```

개발중에 트랜스파일링 한 결과파일은 필요 없으므로 물리적으로 파일은 만들어지지 않지만 웹팩 개발서버는 인지하게 되고 이런 방식으로 개발의 생산성을 꾀한다.

## Step 4 - CLI

<https://cli.angular.io/>

### 1. 설치

```
npm install -g @angular/cli
```

### 명령어 조회

명령어 관련 옵션이 빠르고 자주 변경되고 있다. 버전에 따라 상이 할 수 있으니 직접 확인하기 바란다.

```
ng help
```

#### ng build <options...>

```
Builds your app and places it into the output path (dist/ by default).
aliases: b
--target (String) (Default: development)
  aliases: -t <value>, -dev (--target=development), -prod (--target=production)
--environment (String) (Default: )
  aliases: -e <value>
--output-path (Path) (Default: null)
  aliases: -o <value>
--watch (Boolean) (Default: false)
  aliases: -w
--watcher (String)
--suppress-sizes (Boolean) (Default: false)
--base-href (String) (Default: null)
  aliases: -bh <value>
--aot (Boolean) (Default: false)
--sourcemap (Boolean) (Default: true)
  aliases: -sm
```

#### ng completion

```
Adds autocomplete functionality to `ng` commands and subcommands
```

```
ng doc <keyword>
```

Opens the official Angular documentation for a given keyword.

## ng e2e

Run e2e tests in existing project

## ng generate <blueprint> <options...>

Generates new code from blueprints.

### aliases: g

--dry-run (Boolean) (Default: false)

aliases: -d

--verbose (Boolean) (Default: false)

aliases: -v

--pod (Boolean) (Default: false)

aliases: -p

--classic (Boolean) (Default: false)

aliases: -c

--dummy (Boolean) (Default: false)

aliases: -dum, -id

--in-repo-addon (String) (Default: null)

aliases: --in-repo <value>, -ir <value>

## ng get

Get a value from the configuration.

## ng help

Shows help for the CLI

## ng init <glob-pattern> <options...>

Creates a new angular-cli project in the current folder.

aliases: i

--dry-run (Boolean) (Default: false)

aliases: -d

--verbose (Boolean) (Default: false)

aliases: -v

--link-cli (Boolean) (Default: false)

aliases: -lc

--skip-npm (Boolean) (Default: false)

```
aliases: -sn
--skip-bower (Boolean) (Default: true)
  aliases: -sb
  --name (String) (Default: )
    aliases: -n <value>
  --source-dir (String) (Default: src)
    aliases: -sd <value>
  --style (String) (Default: css)
  --prefix (String) (Default: app)
    aliases: -p <value>
  --mobile (Boolean) (Default: false)
  --routing (Boolean) (Default: false)
  --inline-style (Boolean) (Default: false)
    aliases: -is
  --inline-template (Boolean) (Default: false)
  aliases: -it
```

## ng lint

```
Lints code in existing project
```

## ng new <options...>

```
Creates a new directory and runs ng init in it.
--dry-run (Boolean) (Default: false)
  aliases: -d
--verbose (Boolean) (Default: false)
  aliases: -v
--link-cli (Boolean) (Default: false)
  aliases: -lc
--skip-npm (Boolean) (Default: false)
  aliases: -sn
--skip-bower (Boolean) (Default: true)
  aliases: -sb
--skip-git (Boolean) (Default: false)
  aliases: -sg
--directory (String)
  aliases: -dir <value>
--source-dir (String) (Default: src)
  aliases: -sd <value>
--style (String) (Default: css)
--prefix (String) (Default: app)
```

```
aliases: -p <value>
--mobile (Boolean) (Default: false)
--routing (Boolean) (Default: false)
--inline-style (Boolean) (Default: false)
    aliases: -is
--inline-template (Boolean) (Default: false)
    aliases: -it
```

## ng serve <options...>

```
Builds and serves your app, rebuilding on file changes.
aliases: server, s
--port (Number) (Default: 4200)
    aliases: -p <value>
--host (String) (Default: localhost) Listens only on localhost by default
    aliases: -H <value>
--proxy-config (Path)
    aliases: -pc <value>
--watcher (String) (Default: events)
    aliases: -w <value>
--live-reload (Boolean) (Default: true)
    aliases: -lr
--live-reload-host (String) Defaults to host
    aliases: -lrh <value>
--live-reload-base-url (String) Defaults to baseURL
    aliases: -lrbu <value>
--live-reload-port (Number) (Defaults to port number within [49152...65535])
    aliases: -lrp <value>
--live-reload-live-css (Boolean) (Default: true) Whether to live reload CSS (default true)
--target (String) (Default: development)
    aliases: -t <value>, -dev (--target=development), -prod (--target=production)
--environment (String) (Default: )
    aliases: -e <value>
--ssl (Boolean) (Default: false)
--ssl-key (String) (Default: ssl/server.key)
--ssl-cert (String) (Default: ssl/server.crt)
--aot (Boolean) (Default: false)
--sourcemap (Boolean) (Default: true)
    aliases: -sm
--open (Boolean) (Default: false) Opens the url in default browser
    aliases: -o
```

## **ng set <options...>**

```
Set a value in the configuration.  
--global (Boolean) (Default: false)  
    aliases: -g
```

## **ng test <options...>**

```
Runs your app's test suite.  
aliases: t  
--watch (Boolean) (Default: true)  
    aliases: -w  
--code-coverage (Boolean) (Default: false)  
    aliases: -cc  
--lint (Boolean) (Default: false)  
    aliases: -l  
--single-run (Boolean) (Default: false)  
    aliases: -sr  
--browsers (String)  
--colors (Boolean)  
--log-level (String)  
--port (Number)  
--reporters (String)  
--build (Boolean) (Default: true)  
--sourcemap (Boolean) (Default: true)  
    aliases: -sm
```

## **ng version <options...>**

```
outputs angular-cli version  
aliases: v, --version, -v  
--verbose (Boolean) (Default: false)
```

## **ng e2e**

```
종단 테스트(end to end tests)를 수행한다.
```

옵션 중에 자주 쓸만한 것들이 상당히 존재합니다. 한 번씩 훑어 보는 것을 권해 드립니다.

\* 버전에 따라 내용이 달라질 수 있습니다.

## 2. 새 프로젝트 만들기

```
ng new angular-cli-example  
cd angular-cli-example  
npm start
```

### index.html

```
<!doctype html>  
<html lang="en">  
  <head>  
    <meta charset="utf-8">  
    <title>AngularCliExample</title>  
    <base href="/">  
  
    <meta name="viewport" content="width=device-width, initial-scale=1">  
    <link rel="icon" type="image/x-icon" href="favicon.ico">  
  
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">  
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>  
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>  
    <style type="text/css">  
      .bs-example {  
        margin: 20px;  
      }  
    </style>  
  
  </head>  
  <body>  
    <app-root></app-root>  
  </body>  
</html>
```

부트스트랩을 사용하고자 index.html에 임포트 설정을 한다.

### 홈 컴포넌트 생성

```
ng g component home
```

### app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
```

```

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common'; // BrowserModule 임포트로 생략이 가능하다.
import { HttpClientModule } from '@angular/http';
import { FormsModule } from '@angular/forms';
import { RouterModule, Routes } from '@angular/router';

import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';

const appRoutes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: '**', component: HomeComponent }
];

@NgModule({
  imports: [
    BrowserModule,
    CommonModule, HttpClientModule, FormsModule,
    RouterModule.forRoot(appRoutes)
  ],
  declarations: [
    AppComponent,
    HomeComponent
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

## app.component.html

```

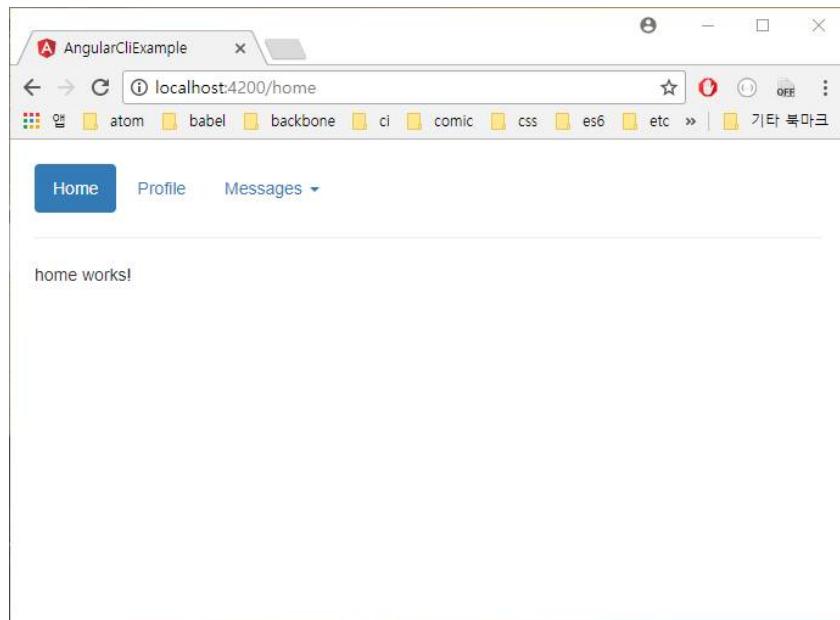
<div class="bs-example">
  <ul class="nav nav-pills">
    <li class="active"><a href="home">Home</a></li>
    <li><a href="#">Profile</a></li>
    <li class="dropdown">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Messages <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a href="#">Inbox</a></li>
        <li><a href="#">Drafts</a></li>
      </ul>
    </li>
  </ul>
</div>

```

```
<li><a href="#">Sent Items</a></li>
<li class="divider"></li>
<li><a href="#">Trash</a></li>
</ul>
</li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>
```

Home 링크를 감싼 `<li>` 태그 내 클래스 설정 `class="active"`가 설정되어 있기 때문에 부트스트랩이 적용되어 디자인적으로 두드러져 보이고 사용자가 현재 어떤 내용을 보고 있는지 직관적으로 파악할 수 있다.

<http://localhost:4200> 주소로 접속



```
ng g component profile
```

### app.module.ts

```
const appRoutes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'profile', component: ProfileComponent },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: '**', component: HomeComponent }
];
```

모듈 파일에 라우팅 설정을 하면 routerLinkActive가 제대로 작동하지 않는 버그가 있었다.

관리 목적 상 라우팅 설정을 별도의 모듈 파일로 분리하는 것이 좋다.

### app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';
import { ProfileComponent } from './profile/profile.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'profile', component: ProfileComponent },
  { path: '**', component: HomeComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

### app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
```

```

import { CommonModule } from '@angular/common'; // BrowserModule 임포트로 생략이 가능하다.
import { HttpClientModule } from '@angular/http';
import { FormsModule } from '@angular/forms';
import { RouterModule, Routes } from '@angular/router';

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { ProfileComponent } from './profile/profile.component';

// const appRoutes: Routes = [
//   { path: 'home', component: HomeComponent },
//   { path: 'profile', component: ProfileComponent },
//   { path: '', redirectTo: 'home', pathMatch: 'full' },
//   { path: '**', component: HomeComponent }
// ];

@NgModule({
  imports: [
    BrowserModule,
    CommonModule, HttpClientModule, FormsModule,
    // RouterModule.forRoot(appRoutes)
    AppRoutingModule
  ],
  declarations: [
    AppComponent,
    HomeComponent,
    ProfileComponent
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

## app.component.html

```

<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li routerLinkActive="active"><a routerLink="profile">Profile</a></li>
    <li class="dropdown">

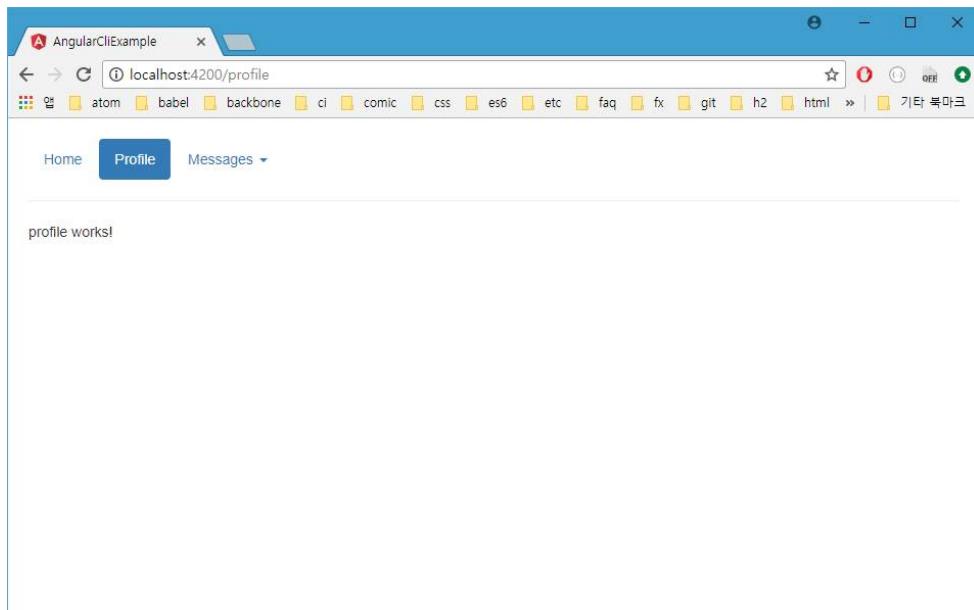
```

```

<a href="#" data-toggle="dropdown" class="dropdown-toggle">Messages <b class="caret"></b></a>
<ul class="dropdown-menu">
    <li><a href="#">Inbox</a></li>
    <li><a href="#">Drafts</a></li>
    <li><a href="#">Sent Items</a></li>
    <li class="divider"></li>
    <li><a href="#">Trash</a></li>
</ul>
</li>
</ul>
<hr/>
<div class="margin">
    <router-outlet></router-outlet>
</div>
</div>

```

앵커 태그의 href 속성 대신 routerLink를 사용한다. href로 설정하면 화면이 리프레쉬가 되지만, routerLink로 설정하면 화면은 그대로 유지하고 화면의 일부분만 갱신된다. routerLinkActive 속성을 사용하여 활성화된 링크가 변경되면 추가적으로 class="active"가 그에 따라 변경되도록 한다.



### 3. 테스트를 위한 빌드

#### 빌드 설정



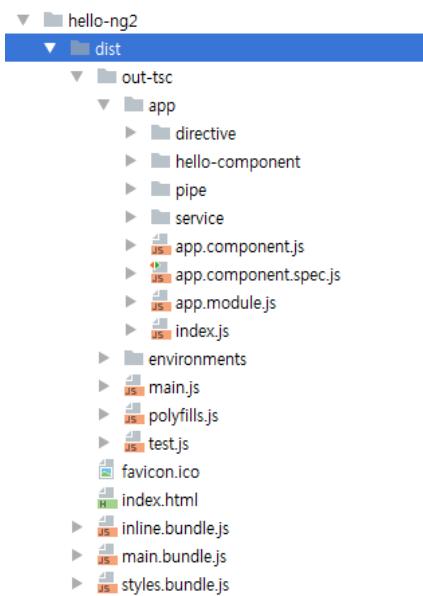
#### environment.ts

```
// The file contents for the current environment will overwrite these during build.  
// The build system defaults to the dev environment which uses `environment.ts`,  
// but if you do  
// `ng build --env=prod` then `environment.prod.ts` will be used instead.  
// The list of which env maps to which file can be found in `angular-cli.json`.  
  
export const environment = {  
  production: false  
};
```

#### 빌드

```
ng build
```

#### 결과



개발 프로젝트에 따라 파일 크기에 다소 차이가 있을 수 있다.

vendor.bundle.js	2.2 MB
polyfills.bundle.js	163 KB
main.bundle.js	13 KB
inline.bundle.js	6 KB
styles.bundle.js	10 KB

보시다시피, 크 사이즈의 vendor.bundle.js 파일이 있다. 프로덕션 환경을 지정하지 않고 ng build를 실행하면 uglifying 및 tree-shaking을 사용하지 않기 때문에 사이즈가 크다.

## 4. 서비스(배포)를 위한 빌드

### 빌드 설정

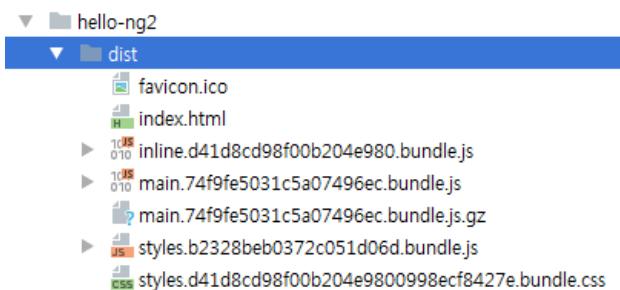
#### environment.prod.ts

```
export const environment = {  
  production: true  
};
```

### 빌드

```
ng build --prod
```

### 결과



테스트를 위한 빌드작업 결과와 배포를 위한 빌드 작업결과의 파일의 크기를 비교해 봅시다.

vendor.bundle.js	352 KB	// Reduced from 2.2 MB
polyfills.bundle.js	57 KB	// Reduced from 163 KB
main.bundle.js	12 KB	// Reduced from 13 KB
inline.bundle.js	2 KB	// Reduced from 6 KB
styles.bundle.js	0 KB	// Reduced from 10 KB

프로덕션 플래그를 추가하면 번들 수가 약 2.4MB에서 352KB로 거의 80% 이상 감소했습니다.

- 파일을 축소하여 원치 않는 공백을 제거합니다.
- 함수 및 변수 이름의 이름을 바꾸어 파일을 Uglify 합니다.
- AoT 컴파일은 런타임에 컴파일 프로세스를 제거하고 대신 빌드 프로세스 중에 컴파일을 수행합니다.

그러면 컴파일 프로세스 관련 코드를 삭제해서 배포할 수 있게 됩니다. 이러한 모든 작업들은 Angular 앱의 파일 크기를 크게 줄여서, 라이브러리 로드 시간을 대폭 줄여줍니다.

중요한 사항 : AoT (Ahead of Time Compilation)에 대해 들어 보셨을 것입니다. 2017년 3월 1일부터 -prod 플래그를 지정하면 자동으로 AoT가 포함됩니다. 이전에는 명시적으로 --aot 플래그를 지정해야 했습니다.

## Deploying your build

이제 앱이 /dist 폴더로 이동할 준비가 되었습니다.

/dist 폴더의 내용을 가져 와서 사이트에 업로드하기만 하면 됩니다. 응용 프로그램은 mysite.com과 같은 루트 공용 폴더에 업로드하는 경우 작동하지만 mysite.com/whatever와 같은 하위 폴더에 있는 경우 빌드 중에 --base-href 플래그를 사용하세요. 그러면 폴더 구조를 기반으로 응용 프로그램이 배치됩니다

## 번들링 결과파일

### inline.bundle.js

이것은 웹팩 로더입니다. 다른 파일을 로드 할 때 필요한 Webpack 유ти리티가 있는 작은 파일입니다.

### vendor.bundle.js

수정 사항이 거의 없거나 전혀 없는 Angular 라이브러리가 포함됩니다. (@angular, RxJS ...)

이것은 빌드 프로세스의 속도를 높이는 것입니다. 또한 많은 사람들이 많이 변경되지 않고 더 이상 캐시 될 수 있는 경우 별도로 보관하는 것이 좋습니다.

개발자의 앱에서 임포트하지 않는 EcmaScript/TypeScript 모듈은 제거됩니다. 이것은 최종 뮤음이 훨씬 작음을 의미합니다. --vendor-chunk 인수를 설정하여 별도의 공급 업체 번들 생성 여부를 명시적으로 제어 할 수 있습니다.

### main.bundle.js

개발자가 추가한 코드

## Step 5 – Component

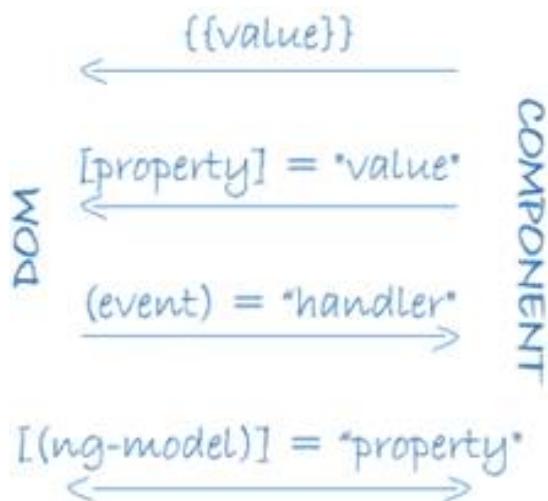
@Component 데코레이터는 바로 아래에있는 클래스를 컴포넌트 클래스로 식별합니다. @Component 데코레이터는 구성 요소와 뷰를 생성하고 표시하는 데 필요한 정보가있는 필수 구성 객체를 사용합니다. 데코레이터라는 용어 대신 널리 알려진 애노테이션이라는 용어를 사용해도 무방합니다.

다음은 가장 유용한 @Component 구성 옵션 중 일부입니다.

- selector : 부모 HTML 에 <hero-list> 태그가있는이 구성 요소의 인스턴스를 만들고 삽입하도록 Angular 에 지시하는 CSS 선택기입니다. 예를 들어 앱의 HTML 에 <hero-list> </ hero-list>가 포함되어 있으면 Angular 는 해당 태그 사이에 HeroListComponent 보기의 인스턴스를 삽입합니다.
- templateUrl : 위에 표시된 구성 요소의 HTML 템플릿의 모듈 기준 주소입니다.
- providers : 구성 요소에 필요한 서비스에 대한 종속성 주입 공급자 배열입니다. Angular 에게 구성 요소의 생성자가 HeroService 를 필요로하므로 표시 할 영웅 목록을 가져올 수 있다고 알려주는 하나의 방법입니다.

## Binding

바인딩은 클래스와 템플릿을 연결하는 기능이다.



클래스의 apple 변수에 들어 있는 값을 입력 엘리먼트로 바인딩하려면 다음과 같이 설정한다.

### 단방향 바인딩

```
template: `<input type="text" [value]="apple">`
```

## 양방향 바인딩

```
template: `<input type="text" [(ngModel)]="apple">`
```

## 단방향 바인딩 예제

```
ng g component interpolation --flat
```

--flat 옵션을 주지 않으면 interpolation 폴더를 만들고 그 밑으로 컴포넌트 파일을 생성한다.

### interpolation.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-interpolation',
  templateUrl: './interpolation.component.html',
  // styleUrls: ['./interpolation.component.css']
  styles: [`.my-italic { font-style: italic; }`]
})
export class InterpolationComponent implements OnInit {
  basket = {
    items: ['apple', 'grapee', 'orange']
  };
  goodbye: string;
  myclass = "my-italic";

  constructor() {
    let x: string = '굿';
    let y: string = '바이';
    this.goodbye = `${x + y}`;
  }

  ngOnInit() {
  }

  hello() {
```

```

        return "hello";
    }

}

```

### **interpolation.component.html**

```

{{1 + 1}}
<br>
{{hello() +" world!"}}
<br>
{{hello() ==="hello"? "YES": "NO"}}
<br>
{{basket.items[0]}}
<br>
{{goodbye}}
<br>
<input type="text" value="{{myclass}}">
<button class="{{myclass}}>{{myclass}}</button>

```

### **app-routing.module.ts**

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';
import { ProfileComponent } from './profile/profile.component';
import { InterpolationComponent } from './binding/interpolation.component';

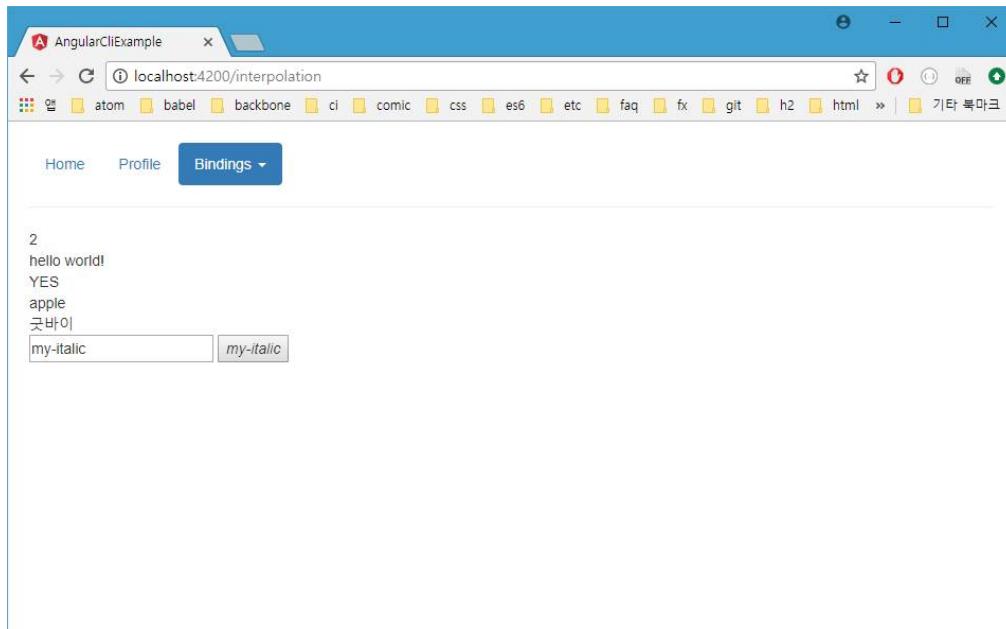
const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'profile', component: ProfileComponent },
  { path: 'interpolation', component: InterpolationComponent },
  { path: '**', component: HomeComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

## app.component.html

```
<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li routerLinkActive="active"><a routerLink="profile">Profile</a></li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Bindings <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="interpolation">Interpolation</a></li>
        <li><a href="#">Drafts</a></li>
        <li><a href="#">Sent Items</a></li>
        <li class="divider"></li>
        <li><a href="#">Trash</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet></router-outlet>
  </div>
</div>
```



```
ng g component binding/oneway --flat
```

### oneway.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-oneway',
  templateUrl: './oneway.component.html',
  styleUrls: ['./oneway.component.css']
})
export class OnewayComponent implements OnInit {
  greeting: string = "hello";

  constructor() { }

  ngOnInit() {
  }
}
```

### oneway.component.html

```
{{greeting}}<br>
<input type="text" [value]="greeting">
<input type="text" bind-value="greeting">
<input type="text" [attr.value]="greeting">
```

### app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';
import { ProfileComponent } from './profile/profile.component';
import { InterpolationComponent } from './binding/interpolation.component';
import { OnewayComponent } from './binding/oneway.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
```

```

{ path: 'home', component: HomeComponent },
{ path: 'profile', component: ProfileComponent },
{ path: 'interpolation', component: InterpolationComponent },
{ path: 'oneway', component: OnewayComponent },
{ path: '**', component: HomeComponent }

];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

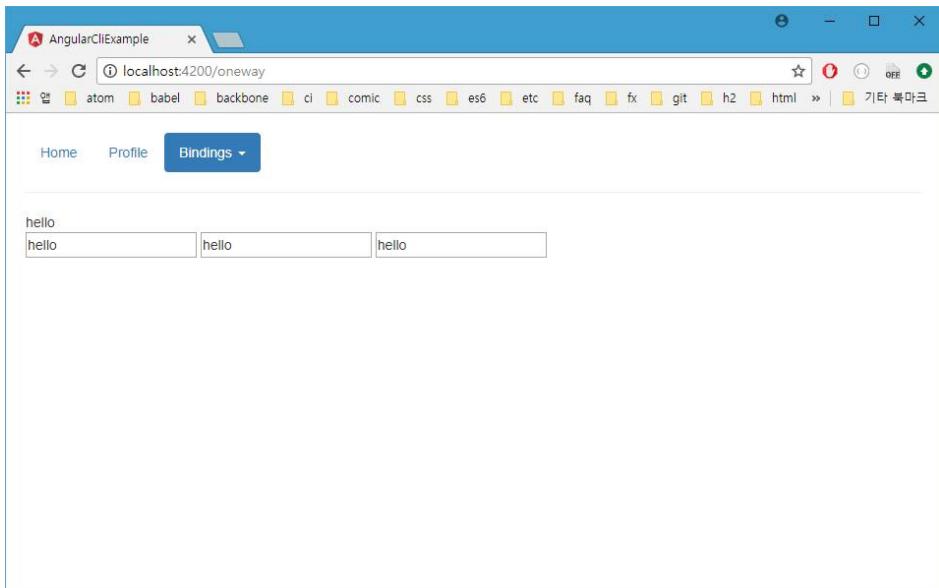
```

### app.component.html

```

<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li routerLinkActive="active"><a routerLink="profile">Profile</a></li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Bindings <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="interpolation">Interpolation</a></li>
        <li><a routerLink="oneway">Oneway</a></li>
        <li><a href="#">Sent Items</a></li>
        <li class="divider"></li>
        <li><a href="#">Trash</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet></router-outlet>
  </div>
</div>

```



```
ng g component binding/contact --flat
```

### contact.component.ts

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'app-contact',
  // templateUrl: './contact.component.html',
  template: `
    <div><input #contactName type="text" placeholder="이름" /></div>
    <div><input #contactTele type="text" placeholder="전화번호" /></div>
    <button (click)="handleClick(contactName.value, contactTele.value)">저장</button>
  `,
  styleUrls: ['./contact.component.css']
})
export class ContactComponent implements OnInit {
  @Output() save: EventEmitter<any> = new EventEmitter();

  constructor() { }

  ngOnInit() {}

  handleClick(name: string, tele: string): void {
    this.save.next({ name: name, telephone: tele });
  }
}
```

저장버튼을 클릭하면 handleClick 메소드가 동작하고 내부에 EventEmitter로 정의한 save 변수를 통해 연락처 컴포넌트를 호출한 app-contact 의 커스텀 속성인 save를 통해 this.save.next에 입력한 정보가 입력한 정보가 \$event 변수로 전달된다.

```
ng g directive binding/myClick
```

### my-click.directive.ts

```
import { Directive, ElementRef, EventEmitter, Output } from '@angular/core';

@Directive({
```

```

    selector: '[myClick]'
  })

export class MyClickDirective {
  toggle = false;

  @Output('myClick') clicks = new EventEmitter<boolean>();

  constructor(el: ElementRef) {
    el.nativeElement.addEventListener('click', (event: Event) => {
      this.toggle = !this.toggle;
      this.clicks.emit(this.toggle ? true : false);
    });
  }
}

```

myClick 지시자를 사용하는 엘리먼트에서 클릭 이벤트가 발생하면 클릭 이벤트에 설정된 내용이 처리되고 this.clicks.emit을 통해 지시자에서 컴포넌트로 값이 전달된다.

**ng g component binding/onewayStatement --flat**

### oneway-statement.component.ts

```

import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-oneway-statement',
  templateUrl: './oneway-statement.component.html',
  styleUrls: ['./oneway-statement.component.css']
})

export class OnewayStatementComponent implements OnInit {
  public msg: string = "버튼을 선택해주세요";
  public msg2: string = "이름과 이메일을 입력해주세요";
  clicked: any = "버튼을 선택해주세요";

  constructor() {}

  ngOnInit() {}

  greetings(msg: string) {

```

```

    this.msg = msg;
}

saveContact(contact) {
    this.msg2 = JSON.stringify(contact);
}

}

```

### **oneway-statement.component.html**

```

<h3>{{msg}}</h3>
<button (click)="greetings('안녕하세요')">안녕하세요 </button>
<button on-click="greetings('환영합니다')">환영합니다 </button>
<br/><br/>

<h3>{{msg2}}</h3>
<app-contact (save)="saveContact($event)"> </app-contact>
<br/><br/>

<h3>{{clicked}}</h3>
<button (myClick)="clicked=$event">클릭 </button>

```

**(click)="greetings('안녕하세요');"**

이벤트 바인딩을 통해 처리한다. 이벤트 바인딩 시 사용할 수 있는 이벤트는 웹 표준 이벤트 규칙을 따른다.

<http://developer.mozilla.org/en-US/docs/Web/Events>

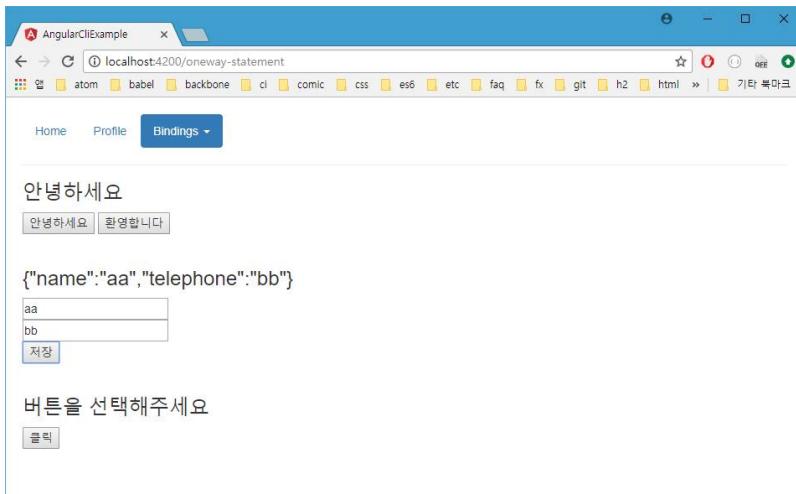
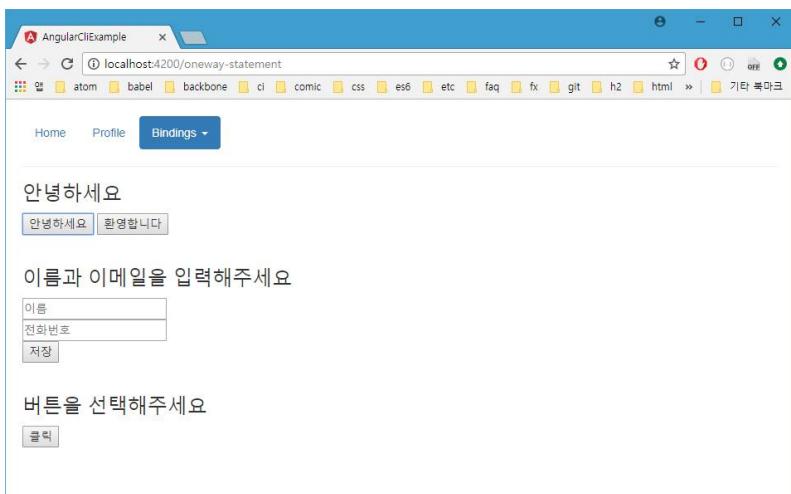
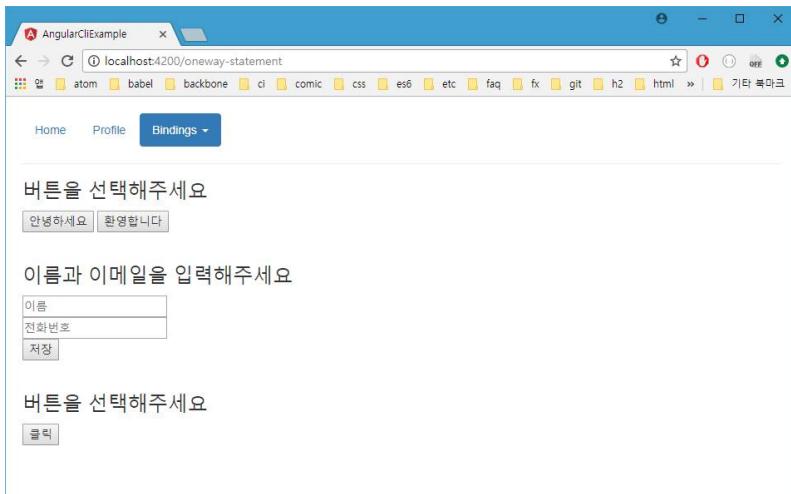
**@Output() save: EventEmitter<any> = new EventEmitter();**

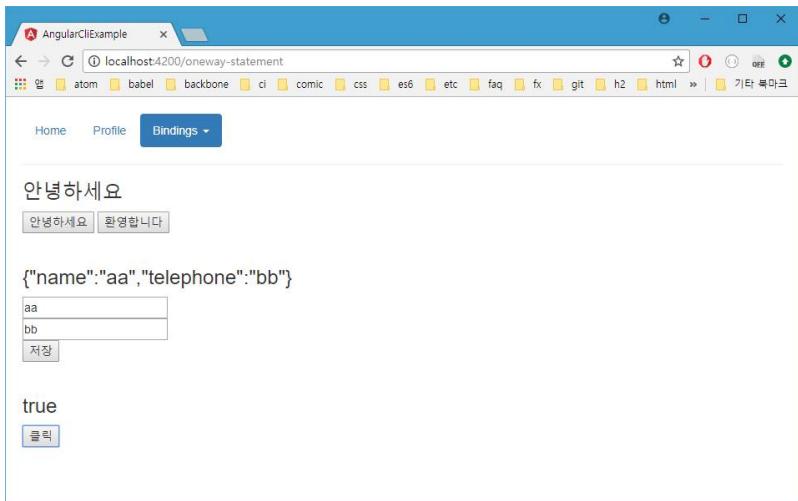
@Output 장식자를 통해 이벤트 명령식인 **(save)="saveContact(\$event)"**를 실행한다.

**@Output('myClick') clicks = new EventEmitter<boolean>();**

@Output 장식자를 통해 이벤트 명령식인 **(myClick)="clicked=\$event"**를 실행한다.

라우팅설정을 추가적으로 작업한 후 결과 화면을 확인하자. 라우팅설정은 앞에서 작업한 내용을 참고로 어느 부분을 해야하는지 파악해 보자.





## 양방향 바인딩 예제

```
ng g component binding/twoWay --flat
```

### twoWay.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-twoWay',
  templateUrl: './twoWay.component.html',
  styleUrls: ['./twoWay.component.css']
})
export class TwoWayComponent implements OnInit {
  city: string = "seoul";
  cities: Object[] = [
    { han: "서울", eng: "seoul" },
    { han: "대전", eng: "daejeon" },
    { han: "대구", eng: "daegu" },
    { han: "부산", eng: "pusan" }
  ];
  constructor() {}

  ngOnInit() {}
}
```

### twoWay.component.html

```
<select (change)="city=$event.target.value">
  <option *ngFor="let obj of cities" [value]="obj.eng" [selected]="city==obj.eng?true:null">
    {{obj.han}}
  </option>
</select>
<br/>
<select [(ngModel)]="city">
  <option *ngFor="let c of cities" [value]="c.eng">{{c.han}}</option>
</select>
```

```

<br/><br/>

<input [value]="city" (input)="city=$event.target.value">
<br/>
<input [(ngModel)]="city">

<br/><br/>

<span *ngFor="let obj of cities">
  <input type="radio" [checked]="(obj.eng==city?true:null)" (click)="city=$event.target.value" [value]="obj.eng"
name="city1">{{obj.han}}
</span>
<br/>
<span *ngFor="let c of cities">
  <input type="radio" [(ngModel)]="city" [value]="c.eng" name="city2">{{c.han}}
</span>

```

### app-routing.module.ts

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';
import { ProfileComponent } from './profile/profile.component';
import { InterpolationComponent } from './binding/interpolation.component';
import { OnewayComponent } from './binding/oneway.component';
import { OnewayStatementComponent } from './binding/oneway-statement.component';
import { TwowayComponent } from './binding/twoway.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'profile', component: ProfileComponent },
  { path: 'interpolation', component: InterpolationComponent },
  { path: 'oneway', component: OnewayComponent },
  { path: 'oneway-statement', component: OnewayStatementComponent },
  { path: 'twoway', component: TwowayComponent },
  { path: '**', component: HomeComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],

```

```

    exports: [RouterModule]
  })
}

export class AppRoutingModule { }

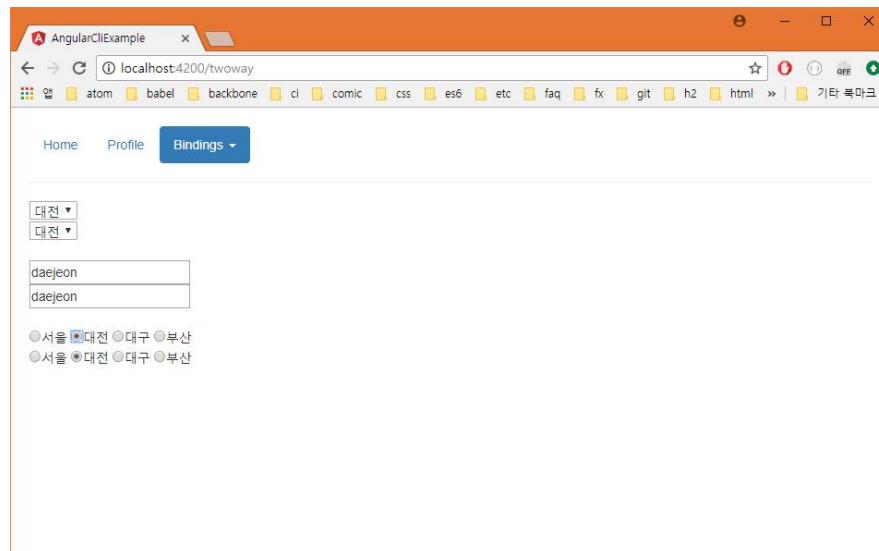
```

### app.component.html

```

<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li routerLinkActive="active"><a routerLink="profile">Profile</a></li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Bindings <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="interpolation">Interpolation</a></li>
        <li><a routerLink="oneway">Oneway</a></li>
        <li><a routerLink="oneway-statement">Oneway Statement</a></li>
        <li class="divider"></li>
        <li><a routerLink="twoway">Twoway</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet></router-outlet>
  </div>
</div>

```



## 따라 해 보기 실습 추천

인터넷에는 좋은 예제가 많이 존재하니 복습 시 여러가지 소스를 분석하세요.

<http://www.concretewebpage.com/angular-2/angular-2-radio-button-and-checkbox-example>

[(ngModel)] 설정의 탄생 배경^^;

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'test2',
  template: `
    <h1>{{title}}</h1>
    <p>{{title}}</p>
    <input value="{{title}}">
    <input [value]="title">
    <input [ngModel]="title">
    <hr>
    <input [value]="title" (input)="title=$event.target.value">
    <input [ngModel]="title" (ngModelChange)="title=$event">
    <hr>
    <input [(ngModel)]="title">
  `,
  styleUrls: ['./test2.component.css']
})
export class Test2Component implements OnInit {
  title = 'Test2Component';

  constructor() { }

  ngOnInit() {
  }
}
```

## 빌트인 지시자

앵귤러가 미리 만들어 놓은 빌트인 디렉티브를 이용하여 바인딩을 처리할 수도 있다.

## 속성 지시자

```
ng g component binding/ngclass -flat
```

### ngclass.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-ngclass',
  templateUrl: './ngclass.component.html',
  styleUrls: ['./ngclass.component.css']
})
export class NgclassComponent implements OnInit {
  public isActive: boolean = false;
  myclass: string = "active";

  constructor() { }

  ngOnInit() {
  }
}
```

### ngclass.component.html

```
<button class="button" [ngClass]="{active: isActive}"  
       (click)="isActive=!isActive;">{{isActive?'활성화':'비 활성화'}}</button>  
<br/>  
  
<button [ngClass]="myclass">버튼 1</button>  
<button [ngClass]=""active"">버튼 2</button>  
<button bind-ngClass="myclass">버튼 3</button>  
<br/>  
  
<button [attr.class]="myclass">버튼 4</button>  
<button [class.active]="true">버튼 5</button>
```

**[ngClass] = "{active: isActive}"**

CSS의 클래스 이름을 더하거나 해제하는 속성 지시자이다. active는 클래스명이다.

#### **attr.class**

attr 속성을 통해 클래스를 추가한다.

#### **class.active**

class 속성을 통해 active 클래스를 추가한다.

### **ngclass.component.css**

```
button {  
    width: 100px;  
    padding: 10px;  
    margin-bottom: 10px;  
    text-align: center;  
    border: 1px dotted #666;  
}  
  
button.active {  
    background-color: #CFD7EB;  
    border: 1px solid #666;  
}
```

### **app.component.html**

```
<div class="bs-example">  
    <ul class="nav nav-pills">  
        <li routerLinkActive="active"><a routerLink="home">Home</a></li>  
        <li routerLinkActive="active"><a routerLink="profile">Profile</a></li>  
        <li class="dropdown" routerLinkActive="active">  
            <a href="#" data-toggle="dropdown" class="dropdown-toggle">Bindings <b class="caret"></b></a>  
            <ul class="dropdown-menu">  
                <li><a routerLink="interpolation">Interpolation</a></li>  
                <li><a routerLink="oneway">Oneway</a></li>  
                <li><a routerLink="oneway-statement">Oneway Statement</a></li>  
                <li class="divider"></li>
```

```

        <li><a routerLink="twoway">Twoway</a></li>
    </ul>
</li>
<li class="dropdown" routerLinkActive="active">
    <a href="#" data-toggle="dropdown" class="dropdown-toggle">Built-in <b class="caret"></b></a>
    <ul class="dropdown-menu">
        <li><a routerLink="ngclass">ngClass</a></li>
    </ul>
</li>
</ul>
<hr/>
<div class="margin">
    <router-outlet></router-outlet>
</div>
</div>

```

### app-routing.module.ts

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

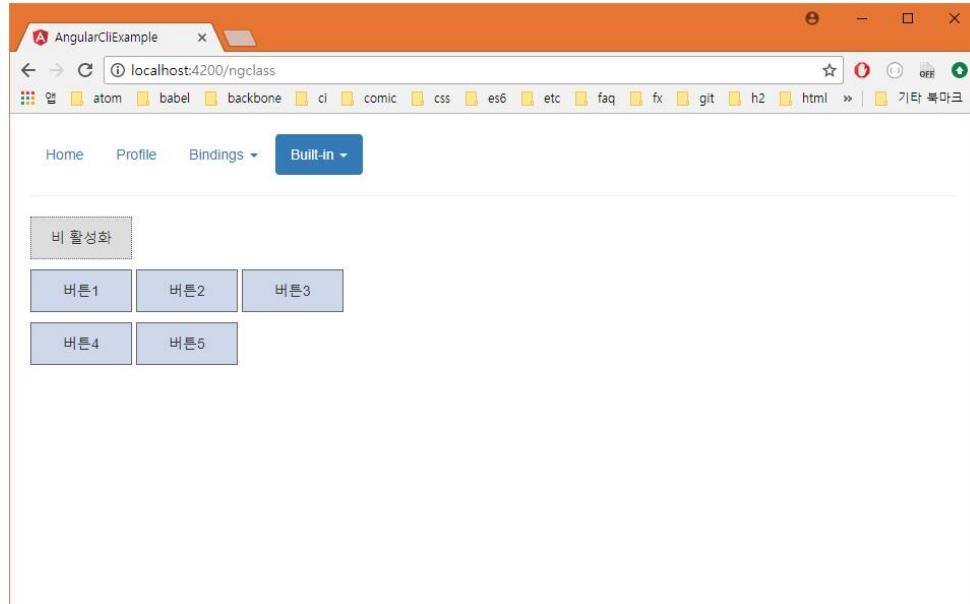
import { HomeComponent } from './home/home.component';
import { ProfileComponent } from './profile/profile.component';
import { InterpolationComponent } from './binding/interpolation.component';
import { OnewayComponent } from './binding/oneway.component';
import { OnewayStatementComponent } from './binding/oneway-statement.component';
import { TwowayComponent } from './binding/twoway.component';
import { NgclassComponent } from './binding/ngclass.component';

const routes: Routes = [
    { path: '', redirectTo: '/home', pathMatch: 'full' },
    { path: 'home', component: HomeComponent },
    { path: 'profile', component: ProfileComponent },
    { path: 'interpolation', component: InterpolationComponent },
    { path: 'oneway', component: OnewayComponent },
    { path: 'oneway-statement', component: OnewayStatementComponent },
    { path: 'twoway', component: TwowayComponent },
    { path: 'ngclass', component: NgclassComponent },
    { path: '**', component: HomeComponent }
];

@NgModule({

```

```
imports: [RouterModule.forRoot(routes)],
exports: [RouterModule]
})
export class AppRoutingModule { }
```



```
ng g component binding/ngstyle --flat
```

### ngstyle.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-ngstyle',
  templateUrl: './ngstyle.component.html',
  styleUrls: ['./ngstyle.component.css']
})
export class NgstyleComponent implements OnInit {
  text = '안녕하세요';
  weight = 'normal';
  style = 'normal';

  constructor() {}

  ngOnInit() {}

  changeWeight($event: any) {
    this.weight = $event.target.checked ? 'bold' : 'normal';
  }
}
```

### ngstyle.component.html

```
<div>
  <input type="text" [(ngModel)]="text">
</div>
<br/>
<div>
  <select [(ngModel)]="style">
    <option value="normal">반듯한 글자</option>
    <option value="italic">기울어진 글자</option>
  </select>
</div>
<br/>
<div>
  <input id="weight" type="checkbox" (change)="changeWeight($event)">
```

```

<label for="weight">볼드체</label>
</div>
<br/>
<h1 [style.font-style]="style" [style.font-weight]="weight">{{text}}</h1>
<h2 [ngStyle]="'{font-style': style, 'font-weight': weight}">{{text}}</h2>

```

ngStyle은 CSS 클래스가 아닌 CSS 스타일 속성과 값을 이용해 설정한다.

#### **(change)="changeWeight(\$event)"**

이벤트 방식으로 함수를 호출해 함수 내부에서 스타일을 결정한다.

이벤트가 발생한 시점에 대한 상태 정보를 전달하기 위해 함수로 \$event 매개변수를 전달한다.

#### **[(ngModel)]="style"**

양방향 바인딩.

### **app-routing.module.ts**

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';
import { ProfileComponent } from './profile/profile.component';
import { InterpolationComponent } from './binding/interpolation.component';
import { OnewayComponent } from './binding/oneway.component';
import { OnewayStatementComponent } from './binding/oneway-statement.component';
import { TwowayComponent } from './binding/twoway.component';
import { NgclassComponent } from './binding/ngclass.component';
import { NgstyleComponent } from './binding/ngstyle.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'profile', component: ProfileComponent },
  { path: 'interpolation', component: InterpolationComponent },
  { path: 'oneway', component: OnewayComponent },
  { path: 'oneway-statement', component: OnewayStatementComponent },
  { path: 'twoway', component: TwowayComponent },
  { path: 'ngclass', component: NgclassComponent },
]

```

```

{ path: 'ngstyle', component: NgstyleComponent },
{ path: '**', component: HomeComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

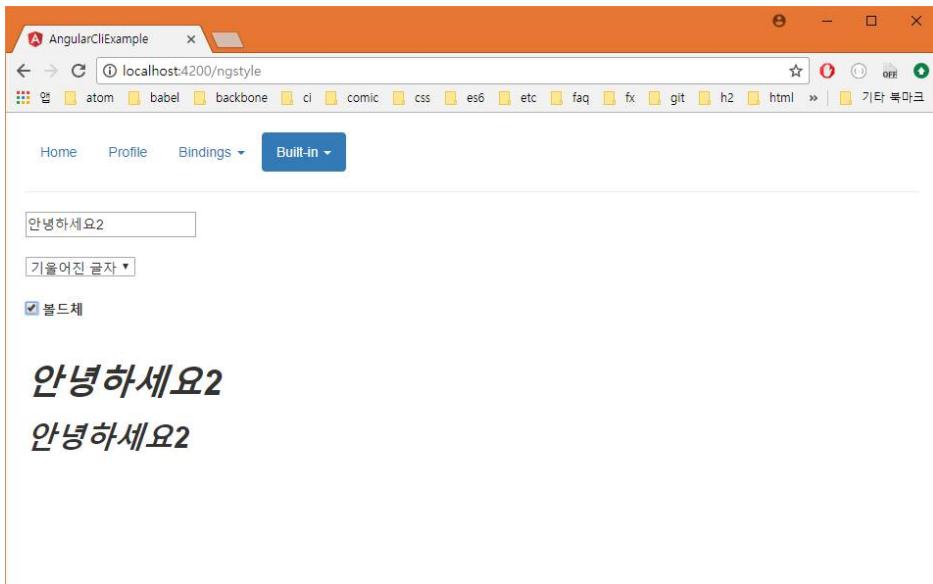
```

### app.component.html

```

<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li routerLinkActive="active"><a routerLink="profile">Profile</a></li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Bindings <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="interpolation">Interpolation</a></li>
        <li><a routerLink="oneway">Oneway</a></li>
        <li><a routerLink="oneway-statement">Oneway Statement</a></li>
        <li class="divider"></li>
        <li><a routerLink="twoway">Twoway</a></li>
      </ul>
    </li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Built-in <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="ngclass">ngClass</a></li>
        <li><a routerLink="ngstyle">ngStyle</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet></router-outlet>
  </div>
</div>

```



## 구조 지시자

```
ng g component binding/ngif --flat
```

### ngif.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-ngif',
  templateUrl: './ngif.component.html',
  styleUrls: ['./ngif.component.css']
})
export class NgifComponent implements OnInit {
  gender = 1;

  constructor() { }

  ngOnInit() {
  }
}
```

### ngif.component.html

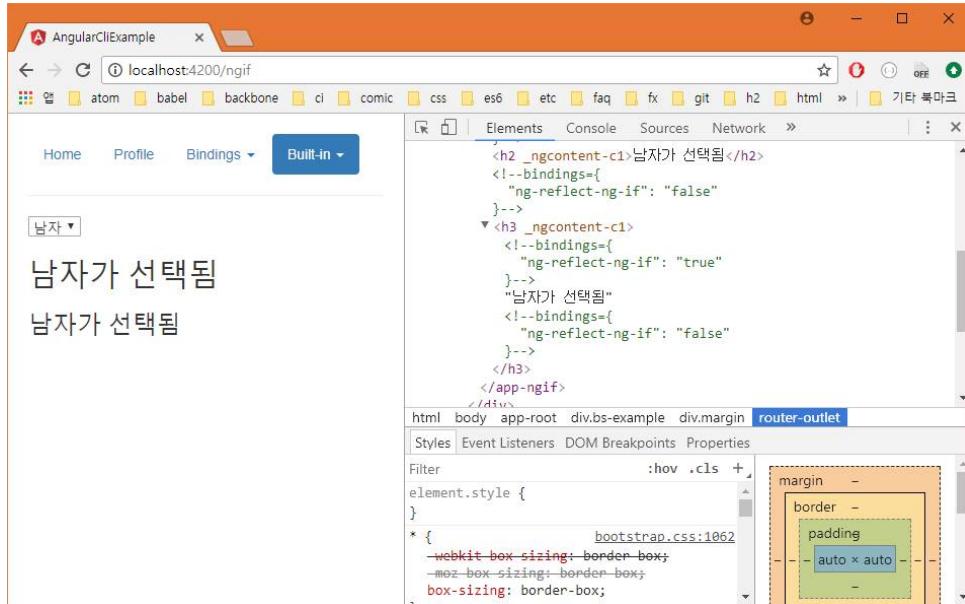
```
<select [(ngModel)]="gender">
  <option value="1">남자</option>
  <option value="2">여자</option>
</select>

<h2 *ngIf="gender == 1">남자가 선택됨</h2>
<h2 *ngIf="gender == 2">여자가 선택됨</h2>

<h3>
  <ng-template [ngIf]="gender == 1">남자가 선택됨</ng-template>
  <ng-template [ngIf]="gender == 2">여자가 선택됨</ng-template>
</h3>
```

ng-template 엘리먼트는 처리가 된 후 사라지는 특징이 있다.

안내: app-routing.module.ts, app.component.html 수정은 앞 부분을 참고해서 작성하세요.



```
ng g component binding/ngswitch --flat
```

## ngswitch.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-ngswitch',
  templateUrl: './ngswitch.component.html',
  styleUrls: ['./ngswitch.component.css']
})
export class NgswitchComponent implements OnInit {
  grade: any;

  constructor() {
    this.grade = 'null';
  }

  ngOnInit() {
  }
}
```

## ngswitch.component.html

```
<div>
  <button (click)="grade='admin'" [style.font-weight]="grade=='admin'?'bold':'normal'">
    운영자</button>
  <button (click)="grade='member'" [style.font-weight]="grade=='member'?'bold':'normal'">
    회원</button>
  <button (click)="grade='guest'" [style.font-weight]="grade=='guest'?'bold':'normal'">
    손님</button>
</div>
<br/>

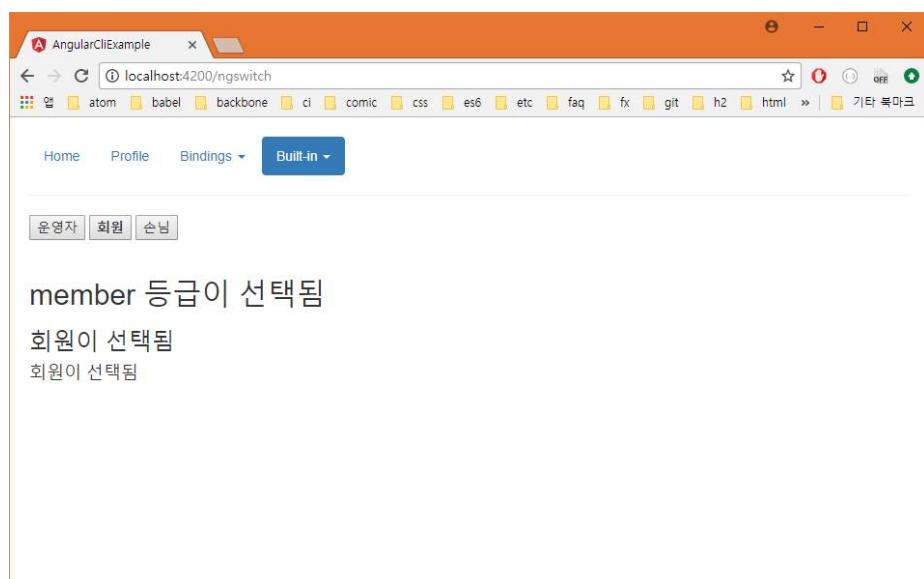
<h2 *ngIf="grade=='null'">회원등급을 선택해주세요</h2>
<h2 *ngIf="grade!='null'">{{grade}} 등급이 선택됨</h2>

<div [ngSwitch]="grade">
  <h3 *ngSwitchCase="">회원등급을 선택해주세요</h3>
  <h3 *ngSwitchCase="admin">운영자가 선택됨</h3>
  <h3 *ngSwitchCase="member">회원이 선택됨</h3>
```

```
<h3 *ngSwitchDefault>손님 선택</h3>
</div>

<h4 [ngSwitch]="grade">
  <ng-template ngSwitchCase="null">회원등급을 선택해주세요</ng-template>
  <ng-template [ngSwitchCase]="'admin'">운영자가 선택됨</ng-template>
  <ng-template [ngSwitchCase]="'member'">회원이 선택됨</ng-template>
  <ng-template ngSwitchDefault>손님 선택</ng-template>
</h4>
```

안내: app-routing.module.ts, app.component.html 설정은 앞 부분을 참고해서 작성하세요.



```
ng g component binding/ngfor --flat
```

### ngfor.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-ngfor',
  templateUrl: './ngfor.component.html',
  styleUrls: ['./ngfor.component.css']
})
export class NgforComponent implements OnInit {
  items: Object[] = [];

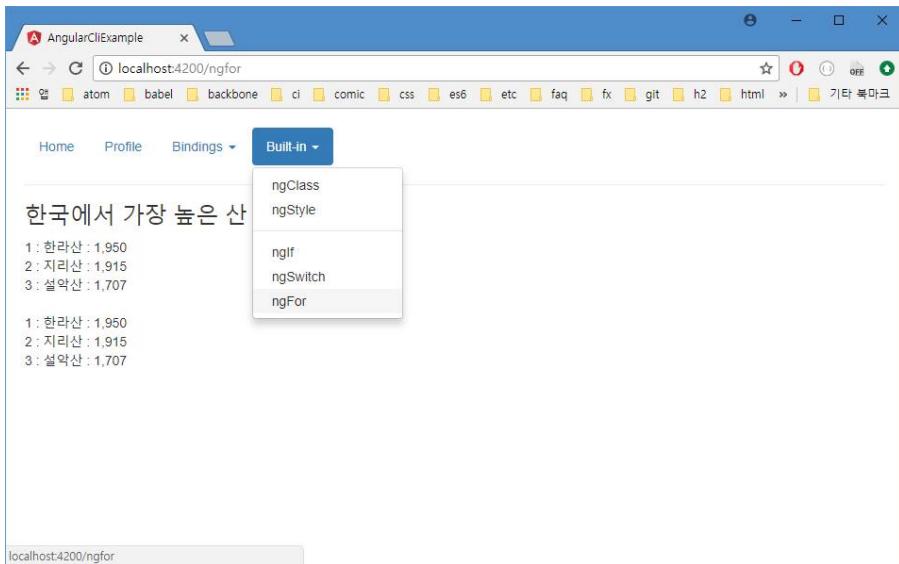
  constructor() {
    this.items.push({ 'title': '한라산', 'height': '1950' });
    this.items.push({ 'title': '지리산', 'height': '1915' });
    this.items.push({ 'title': '설악산', 'height': '1707' });
  }

  ngOnInit() {
  }
}
```

### ngfor.component.html

```
<h3>한국에서 가장 높은 산 TOP3</h3>
<div *ngFor="let item of items; let i = index">
  {{ i+1 }} : {{item.title}} : {{item.height | number}}
</div>
<br/>

<div>
  <ng-template ngFor let-item let-i="index" [ngForOf]="items">
    {{ i+1 }} : {{item.title}} : {{item.height | number}}<br/>
  </ng-template>
</div>
```



## 템플릿 참조변수

템플릿 내에서 정의하고 템플릿 내부에서만 사용할 수 있다. 간단한 상태 전달이 필요한 경우에 이용한다.

```
ng g component binding/ref --flat
```

### ref.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-ref',
  templateUrl: './ref.component.html',
  styleUrls: ['./ref.component.css']
})
export class RefComponent implements OnInit {
  init = { num1: 10, num2: 20 };

  constructor() {}

  ngOnInit() {
  }
}
```

### ref.component.html

```
<input #num1 type="number" value="{{init.num1}}>
+
<input #num2 type="number" value="{{init.num2}}" (input)="0">
=
{{num1.valueAsNumber + num2.valueAsNumber}}
<br/>

<input ref-number1 type="number" value="{{init.num1}}" (input)="init.num1 = $event.target.value">
+
<input ref-number2 type="number" value="{{init.num2}}" (input)="init.num2 = number2.value">
=
{{number1.valueAsNumber + number2.valueAsNumber}}
<br/><br/>
```

```
 {{init | json}}
```

(input)="0"

입력 이벤트를 감지하여 바로 {{ }} 부분에 갠신이 이루어지게 한다. 숫자 0은 특별한 의미를 갖지 않는다.

(keyup) 또는 (keydown) 이벤트로 대체할 수 있다. (keydown.enter) 설정은 엔터키 입력 시에만 작동한다.

(input)="init.num1 = \$event.target.value"

입력 값을 컴포넌트의 변수에 바로 반영시킨다.

### valueAsNumber

HTML5의 입력 엘리먼트의 속성으로 숫자일 경우 출력하고 그렇지 않으면 NaN 값을 출력한다.

## app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';
import { ProfileComponent } from './profile/profile.component';
import { InterpolationComponent } from './binding/interpolation.component';
import { OnewayComponent } from './binding/oneway.component';
import { OnewayStatementComponent } from './binding/oneway-statement.component';
import { TwowayComponent } from './binding/twoway.component';
import { NgclassComponent } from './binding/ngclass.component';
import { NgstyleComponent } from './binding/ngstyle.component';
import { NgifComponent } from './binding/ngif.component';
import { NgswitchComponent } from './binding/ngswitch.component';
import { NgforComponent } from './binding/ngfor.component';
import { RefComponent } from './binding/ref.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'profile', component: ProfileComponent },
  { path: 'interpolation', component: InterpolationComponent },
  { path: 'oneway', component: OnewayComponent },
  { path: 'oneway-statement', component: OnewayStatementComponent },
```

```

    { path: 'twoWay', component: TwoWayComponent },
    { path: 'ngClass', component: NgClassComponent },
    { path: 'ngStyle', component: NgStyleComponent },
    { path: 'ngIf', component: NgIfComponent },
    { path: 'ngSwitch', component: NgSwitchComponent },
    { path: 'ngFor', component: NgForComponent },
    { path: 'ref', component: RefComponent },
    { path: '**', component: HomeComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

### app.component.html

```

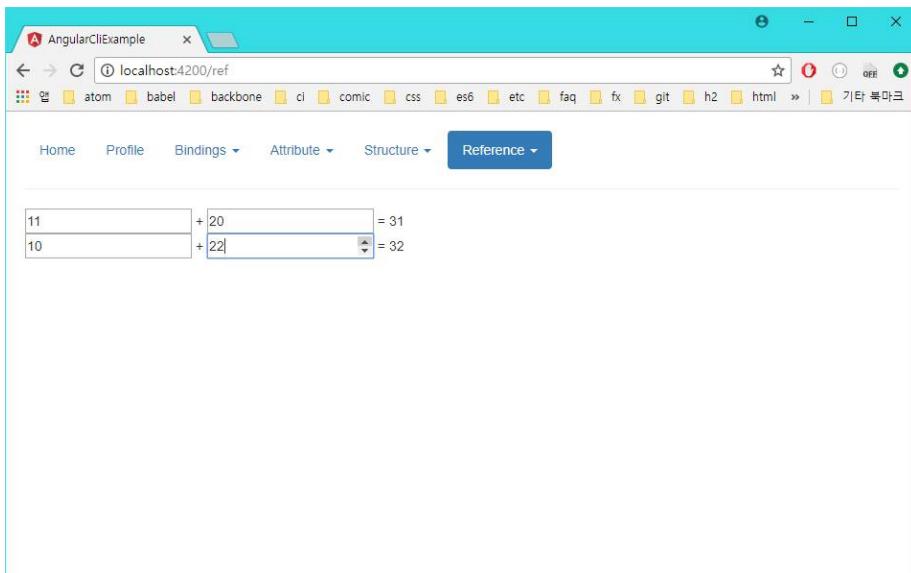
<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li routerLinkActive="active"><a routerLink="profile">Profile</a></li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Bindings <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="interpolation">Interpolation</a></li>
        <li><a routerLink="oneway">Oneway</a></li>
        <li><a routerLink="oneway-statement">Oneway Statement</a></li>
        <li class="divider"></li>
        <li><a routerLink="twoWay">TwoWay</a></li>
      </ul>
    </li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Attribute <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="ngClass">ngClass</a></li>
        <li><a routerLink="ngStyle">ngStyle</a></li>
      </ul>
    </li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Structure <b class="caret"></b></a>
      <ul class="dropdown-menu">

```

```

<li><a routerLink="ngif">ngIf</a></li>
<li><a routerLink="ngswitch">ngSwitch</a></li>
<li><a routerLink="ngfor">ngFor</a></li>
</ul>
</li>
<li class="dropdown" routerLinkActive="active">
  <a href="#" data-toggle="dropdown" class="dropdown-toggle">Reference <b class="caret"></b></a>
  <ul class="dropdown-menu">
    <li><a routerLink="ref">ref</a></li>
  </ul>
</li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>

```



## Step 6 – Service

```
ng new angular-basic-parts --routing=true
```

### index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>AngularBasicParts</title>
    <base href="/">

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
    <style type="text/css">
      .bs-example {
        margin: 20px;
      }
    </style>

  </head>
  <body>
    <app-root></app-root>
  </body>
</html>
```

```
ng g component home
```

### app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/http';
```

```

import { FormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';

@NgModule({
  imports: [
    BrowserModule,
    HttpClientModule,
    FormsModule,
    AppRoutingModule,
  ],
  declarations: [
    AppComponent,
    HomeComponent
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

### app.component.html

```

<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li><a href="#">Profile</a></li>
    <li class="dropdown">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Messages <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a href="#">Inbox</a></li>
        <li><a href="#">Drafts</a></li>
        <li><a href="#">Sent Items</a></li>
        <li class="divider"></li>
        <li><a href="#">Trash</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet> </router-outlet>
  </div>
</div>

```

```
</div>  
</div>
```

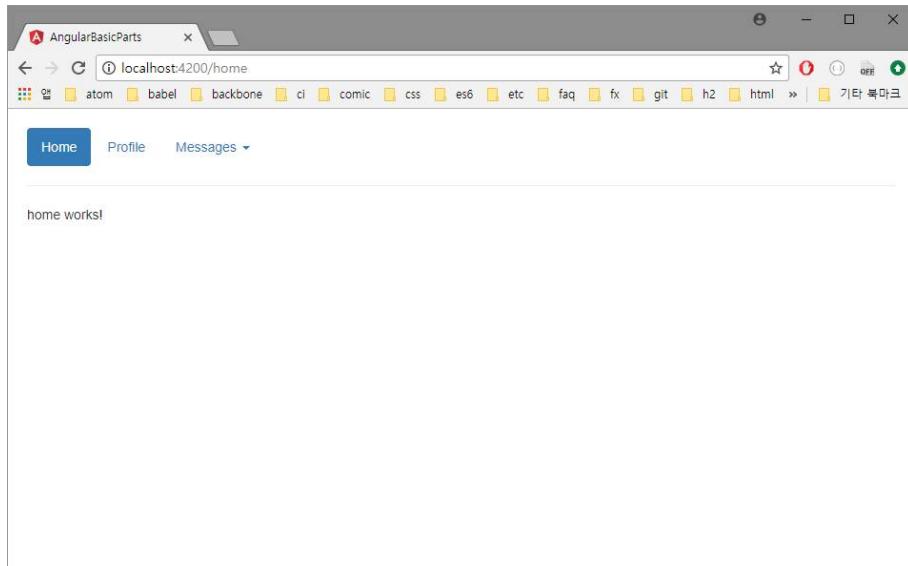
### app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: '**', component: HomeComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```



## 서비스 추가

```
cd src/app && mkdir service && cd ../../
ng g service service/hello
```

서비스 클래스를 별도의 폴더 밑에 배치하고 싶다면 미리 해당 폴더를 만들어야 한다.

```
installing service
create src\app\service\hello.service.spec.ts
create src\app\service\hello.service.ts
WARNING Service is generated but not provided, it must be provided to be used
```

서비스는 모듈에 자동적으로 추가되지 않으므로 개발자가 직접 설정작업을 수행해야 한다.

### app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/http';
import { FormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';

import { HelloService } from './service/hello.service';

@NgModule({
  imports: [
    BrowserModule,
    HttpClientModule,
    FormsModule,
    AppRoutingModule,
  ],
  declarations: [
    AppComponent,
    HomeComponent
  ],
  providers: [HelloService],
  bootstrap: [AppComponent]
})
```

```
export class AppModule { }
```

### hello.service.ts

```
import { Injectable } from '@angular/core';

@Injectable()
export class HelloService {
  helloTitle = 'This is HelloService's Title.';

  constructor() { }

  getTitle(){
    return this.helloTitle;
  }

  setTitle(title){
    this.helloTitle = title;
  }
}
```

### home.component.ts

```
import { Component, OnInit } from '@angular/core';
import { HelloService } from './service/hello.service';

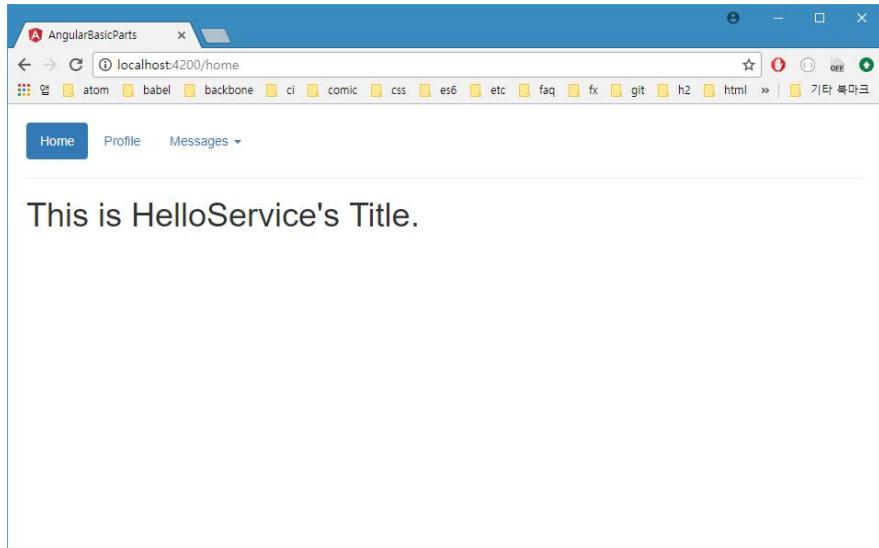
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {
  homeTitle;

  constructor(private helloService: HelloService) {}

  ngOnInit() {
    this.homeTitle = this.helloService.getTitle();
  }
}
```

## home.component.html

```
<h1>{{homeTitle}}</h1>
```



## Step 7 – Pipe

파이프는 컨텐츠를 화면에 표시하기 직전에 적용하는 데이터가공 기능이다.

### 빌트인 파이프

앵귤러가 미리 만들어 놓은 파이프는 common 모듈을 임포트하면 바로 이용할 수 있다.

```
ng g component built-in-pipe
```

#### built-in-pipe.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-built-in-pipe',
  templateUrl: './built-in-pipe.component.html',
  styleUrls: ['./built-in-pipe.component.css']
})
export class BuiltInPipeComponent implements OnInit {
  str = "abcDEF";
  num = 123456.123456;
  date = new Date();
  money = 1000000;
  json = {
    info: { name: '사용자 1', age: 20 },
    list: [
      { name: '사용자 1', age: 20 },
      { name: '사용자 2', age: 20 }
    ]
  };
  items = ['AA', 'BB', 'CC', 'DD'];

  constructor() {}

  ngOnInit() {
  }
}
```

## **built-in-pipe.component.html**

```
<h3>문자열</h3>
string: <label>{{str}}</label> <br>
string | uppercase: <label>{{str | uppercase}}</label> <br>
string | lowercase: <label>{{str | lowercase}}</label> <br>

<h3>슬라이스</h3>
string: <label>{{str}}</label> <br>
string | slice:0:3: <label>{{str | slice:0:3}}</label> <br>
string | slice:3:5: <label>{{str | slice:3:5}}</label> <br>

<h3>숫자</h3>
number: <label>{{num}}</label> <br>
number | number: <label>{{num | number}}</label> <br>
number | number:'3-4': <label>{{num | number:'3-4'}}</label> <br>
number | number:'10.0-3': <label>{{num | number:'10.0-3'}}</label> <br>

<h3>퍼센트</h3>
number: <label>{{num}}</label> <br>
number | percent: <label>{{num | percent}}</label> <br>
number | percent:'2.1-2': <label>{{num | percent:'2.1-2'}}</label> <br>

<h3>날짜</h3>
date: <label>{{date}}</label> <br>
date | date: <label>{{date | date}}</label> <br>
date | date:'HH:mm': <label>{{date | date:'HH:mm'}}</label> <br>
date | date:'fullDate': <label>{{date | date:"fullDate"}}</label> <br>
date | date:'yyyy/MM/dd': <label>{{date | date:"yyyy/MM/dd"}}</label> <br>

<h3>통화</h3>
money: <label>{{money}}</label> <br>
money | currency: <label>{{money | currency}}</label> <br>
money | currency:'USD': <label>{{money | currency:'USD'}}</label> <br>
money | currency:'USD':true: <label>{{money | currency:'USD':true}}</label> <br>
money | currency:'KRW': <label>{{money | currency:'KRW'}}</label> <br>
money | currency:'KRW':true: <label>{{money | currency:'KRW':true}}</label> <br>
money | currency:'EUR':false:'3.2-2': <label>{{money | currency:'EUR':false:'3.2-2'}}</label> <br>

<h3>JSON</h3>
json: <label>{{json}}</label> <br>
json: <label>{{json|json}}</label> <br>

<pre>
  <label>{{json|json}}</label>
</pre>

<h3>슬라이스 2</h3>
<ul>
  <li *ngFor="let item of items">
    {{ item }}
  </li>
</ul>
```

```
</li>
</ul>
<br>
<ul>
  <li *ngFor="let item of items | slice:1:3">
    {{ item }}
  </li>
</ul>
```

{{num | number:'10.0-3'}}

10 : 최소 개수

0 : 최소 개수, 3 : 최대 개수

### built-in-pipe.component.css

```
label {
  color: red;
}
```

### app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

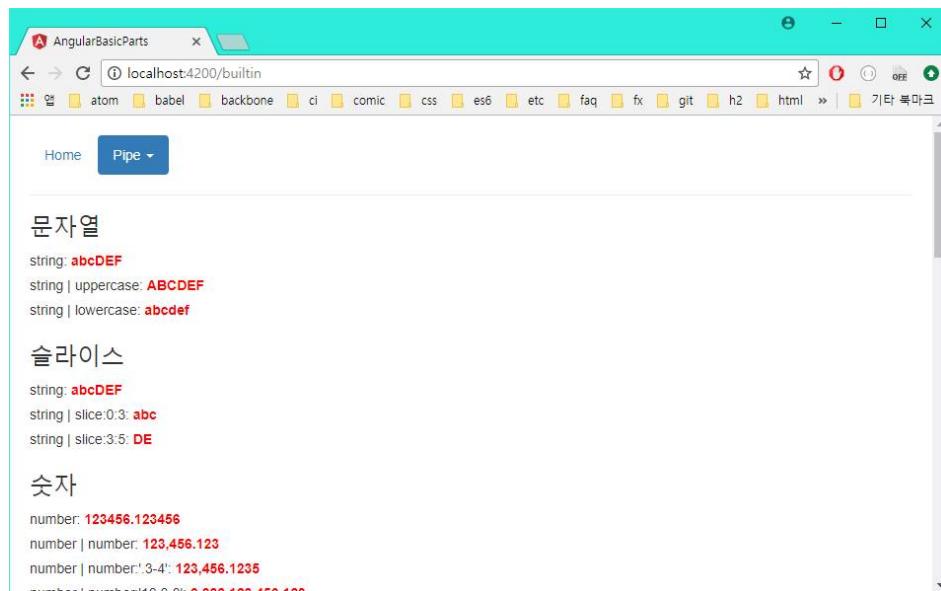
import { HomeComponent } from './home/home.component';
import { BuiltInPipeComponent } from './built-in-pipe/built-in-pipe.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'builtin', component: BuiltInPipeComponent },
  { path: '**', component: HomeComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

## app.component.html

```
<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Pipe <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="builtin">Built-in Pipe</a></li>
        <li><a routerLink="#">#</a></li>
        <li><a routerLink="#">#</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet></router-outlet>
  </div>
</div>
```



The screenshot shows a browser window titled "AngularBasicParts" with the URL "localhost:4200/builtin". The page content displays the following Angular code:

```
},
"list": [
  {
    "name": "사용자1",
    "age": 20
  },
  {
    "name": "사용자2",
    "age": 20
  }
]
```

Below the code, there is a heading "슬라이스2" followed by a bulleted list:

- AA
- BB
- CC
- DD

- BB
- CC

## 커스텀 파이프

```
cd src/app && mkdir pipe && cd ../../
```

```
ng g pipe pipe/my-date
```

### my-date.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'myDate'
})
export class MyDatePipe implements PipeTransform {
  // {{str | myDate:'/'}} 에서
  // str : 첫 번째 파라미터
  // '/' : 두 번째 파라미터
  transform(value: any, exponent?: any): any {
    if (!exponent) {
      exponent = '-';
    }
    if (value.length == 8) {
      return value.substring(0, 4) + exponent +
        value.substring(4, 6) + exponent +
        value.substring(6, 8);
    } else {
      return value;
    }
  }
}
```

exponent?: any

? : 이 파라미터는 옵셔널하다. 주어도 되고 주지 않아도 된다.

```
ng g component custom-pipe
```

### custom-pipe.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-custom-pipe',
  templateUrl: './custom-pipe.component.html',
  styleUrls: ['./custom-pipe.component.css']
})
export class CustomPipeComponent implements OnInit {
  str = "20170311";

  constructor() {}

  ngOnInit() {
  }
}
```

### custom-pipe.component.html

```
<h3>커스텀</h3>
str: {{str}} <br>
str | myDate: {{str | myDate}} <br>
str | myDate:'/': {{str | myDate:'/'}} <br>
```

### app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';
import { BuiltInPipeComponent } from './built-in-pipe/built-in-pipe.component';
import { CustomPipeComponent } from './custom-pipe/custom-pipe.component';

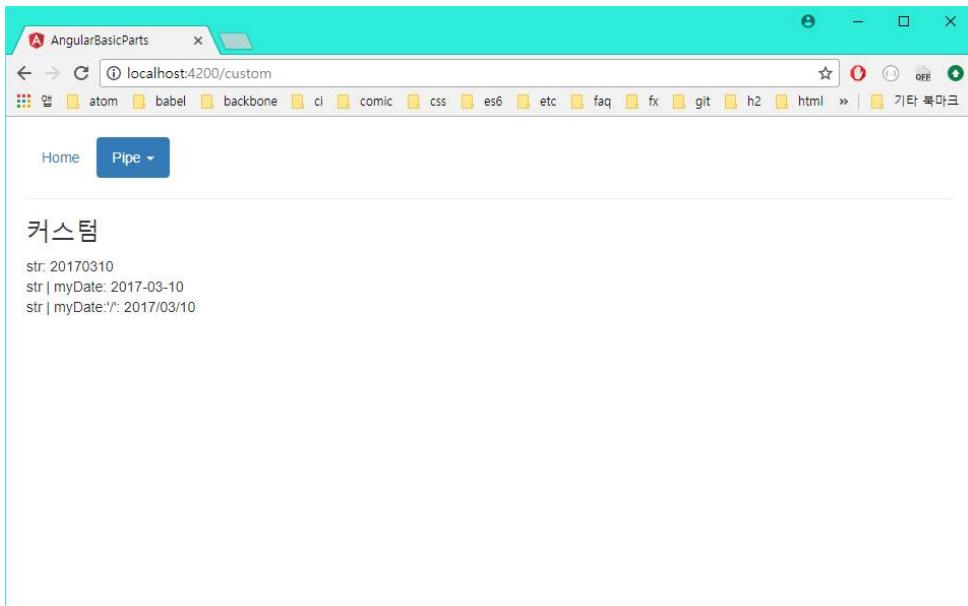
const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'builtin', component: BuiltInPipeComponent },
  { path: 'custom', component: CustomPipeComponent },
  { path: '**', component: HomeComponent }
```

```
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

### app.component.html

```
<div class="bs-example">
<ul class="nav nav-pills">
  <li routerLinkActive="active"><a routerLink="home">Home</a></li>
  <li class="dropdown" routerLinkActive="active">
    <a href="#" data-toggle="dropdown" class="dropdown-toggle">Pipe <b class="caret"></b></a>
    <ul class="dropdown-menu">
      <li><a routerLink="builtin">Buit-in Pipe</a></li>
      <li><a routerLink="custom">Custom Pipe</a></li>
      <li><a routerLink="#">#</a></li>
      <li class="divider"></li>
      <li><a routerLink="#">#</a></li>
    </ul>
  </li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>
```



```
ng g pipe pipe/my-reverse
```

### my-reverse.pipe.ts

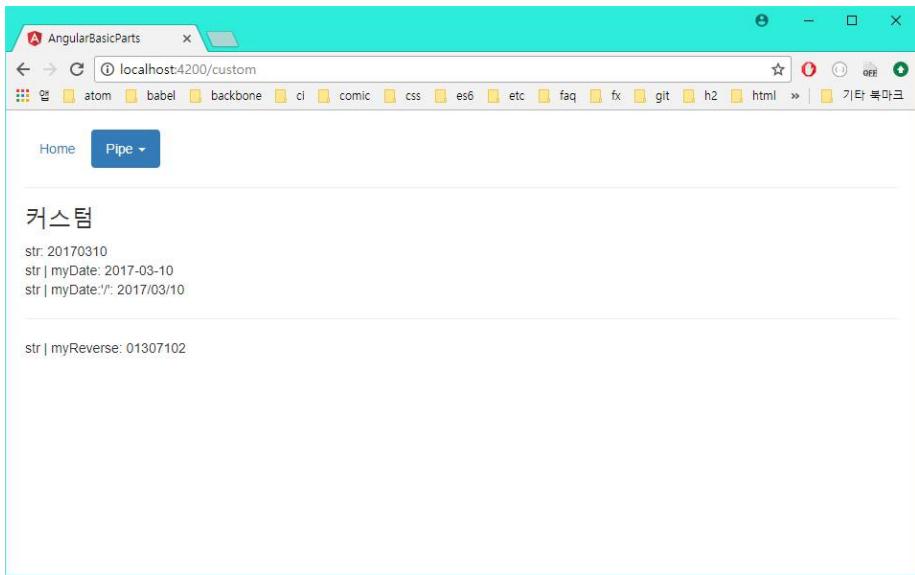
```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'myReverse'
})
export class MyReversePipe implements PipeTransform {

  transform(value: string): string {
    if(value){
      return value.split('').reverse().join('');
    }
    return null;
  }
}
```

### custom-pipe.component.html

```
<h3>커스텀</h3>
str: {{str}} <br>
str | myDate: {{str | myDate}} <br>
str | myDate:'/: {{str | myDate:'/'}} <br>
<hr>
str | myReverse: {{str | myReverse}} <br>
```



## Step 8 - Directive

컴포넌트는 뷰가 있지만 디렉티브는 뷰를 갖고 있지 않다. 디렉티브는 이미 존재하는 엘리먼트에 특정 행동을 추가하는 역할이다. 이해를 돋기위해서 부가적인 설명을 하자면 컴포넌트는 디렉티브에 화면을 추가했다고 생각하자.

### 커스텀 디렉티브

```
cd src/app && mkdir directive && cd ../../
```

```
ng g directive directive/my-hidden
```

#### my-hidden.directive.ts

```
import { Directive, ElementRef, Renderer } from '@angular/core';

@Directive({
  selector: '[myHidden]'
})
export class MyHiddenDirective {

  constructor(el: ElementRef, renderer: Renderer) {
    renderer.setStyle(el.nativeElement, 'display', 'none');
  }
}
```

```
ng g component custom-directive
```

#### custom-directive.component.html

```
<h1>Welcome</h1>
<h1 myHidden>Hidden Welcome</h1>
```

커스텀 디렉티브 myHidden이 설정된 엘리먼트는 style="display: none;" 속성이 설정되어 보이지 않는다.

## app.component.html

```
<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Pipe <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="builtin">Buit-in Pipe</a></li>
        <li><a routerLink="custom">Custom Pipe</a></li>
        <li><a routerLink="#">#</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
      </ul>
    </li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Directive <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="custom-directive">Custom Directive</a></li>
        <li><a routerLink="#">#</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet></router-outlet>
  </div>
</div>
```

## app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

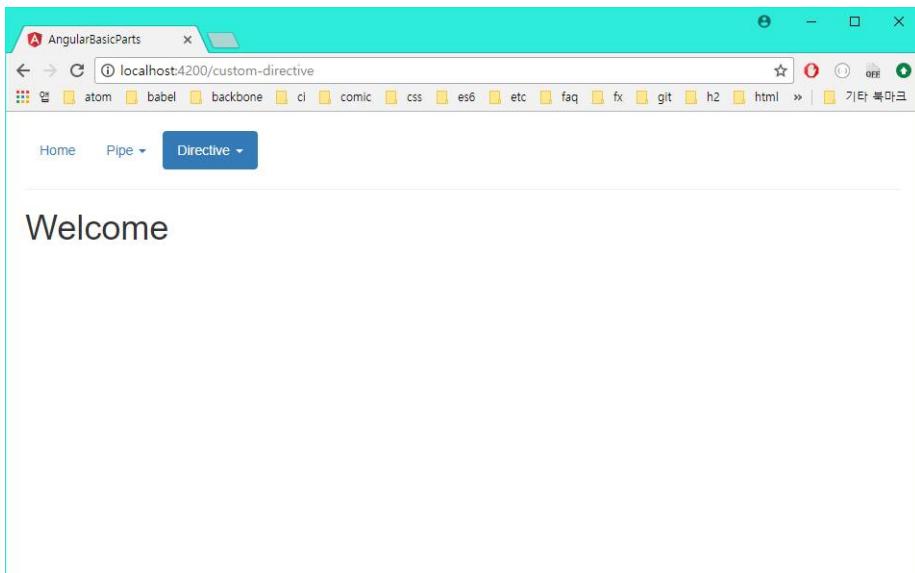
import { HomeComponent } from './home/home.component';
import { BuiltInPipeComponent } from './built-in-pipe/built-in-pipe.component';
import { CustomPipeComponent } from './custom-pipe/custom-pipe.component';
import { CustomDirectiveComponent } from './custom-directive/custom-directive.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
```

```
{ path: 'home', component: HomeComponent },
{ path: 'builtin', component: BuiltInPipeComponent },
{ path: 'custom', component: CustomPipeComponent },
{ path: 'custom-directive', component: CustomDirectiveComponent },
{ path: '**', component: HomeComponent }

};

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```



```
ng g directive directive/my-hidden2
```

### my-hidden2.directive.ts

```
import { Directive, Input, ElementRef, Renderer, DoCheck } from '@angular/core';

@Directive({
  selector: '[myHidden2]'
})
export class MyHidden2Directive implements DoCheck{

  constructor(public el: ElementRef, public renderer: Renderer) {
    console.log('MyHidden2Directive # constructor() called.')
  }

  @Input() myHidden2: boolean;

  ngDoCheck() {
    if (this.myHidden2) {
      this.renderer.setStyle(this.el.nativeElement, 'display', 'none');
    } else {
      this.renderer.setStyle(this.el.nativeElement, 'display', '');
    }
  }
}
```

컴포넌트의 상태값이 변경되어 디렉티브에게 전달될 때 마다 로직을 수행하려면 ngOnInit 가 아니라 ngDoCheck 함수에 로직을 배치해야 한다. ngOnInit는 객체가 된 후 한 번만 기동한다.

```
ng g component custom-directive2
```

### custom-directive2.component.ts

```
import { Component, OnInit, DoCheck } from '@angular/core';

@Component({
  selector: 'app-custom-directive2',
  templateUrl: './custom-directive2.component.html',
  styleUrls: ['./custom-directive2.component.css']
})
```

```

export class CustomDirective2Component implements OnInit, DoCheck {
  isShow = true;

  constructor() { }

  ngOnInit() {
  }

  ngDoCheck() {
    console.log(this.isShow);
  }
}

```

### custom-directive2.component.html

```

<h1>Welcome</h1>
<button type="button" (click)="isShow=!isShow">toggle</button>
<h1 [myHidden2]="isShow">Hidden Welcome</h1>

```

[myHidden2]="isShow"

isShow에 대괄호를 같이 표기하면 객체로 전달되니 주의가 필요하다.

### app.component.html

```

<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Pipe <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="builtin">Built-in Pipe</a></li>
        <li><a routerLink="custom">Custom Pipe</a></li>
        <li><a routerLink="#">#</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
      </ul>
    </li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Directive <b class="caret"></b></a>
      <ul class="dropdown-menu">

```

```

<li><a routerLink="custom-directive">Custom Directive</a></li>
<li><a routerLink="custom-directive2">Custom Directive2</a></li>
<li class="divider"></li>
<li><a routerLink="#">#</a></li>
</ul>
</li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>

```

### app-routing.module.ts

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

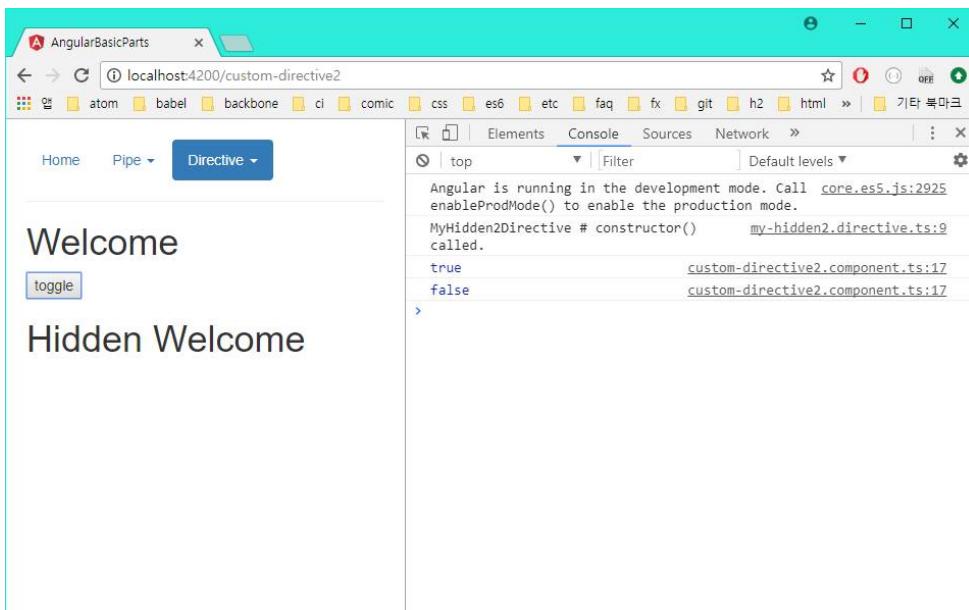
import { HomeComponent } from './home/home.component';
import { BuiltInPipeComponent } from './built-in-pipe/built-in-pipe.component';
import { CustomPipeComponent } from './custom-pipe/custom-pipe.component';
import { CustomDirectiveComponent } from './custom-directive/custom-directive.component';
import { CustomDirective2Component } from './custom-directive2/custom-directive2.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'builtin', component: BuiltInPipeComponent },
  { path: 'custom', component: CustomPipeComponent },
  { path: 'custom-directive', component: CustomDirectiveComponent },
  { path: 'custom-directive2', component: CustomDirective2Component },
  { path: '**', component: HomeComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

테스트 해 보면 toggle 버튼을 누를 때 마다 문자열이 나타났다 사라졌다를 반복한다.



```
ng g directive directive/my-highlight
```

### my-highlight.directive.ts

```
import {Directive, ElementRef, Renderer, HostListener} from '@angular/core';

@Directive({
  selector: '[myHighlight]'
})
export class MyHighlightDirective {

  constructor(private el: ElementRef, private renderer: Renderer) { }

  @HostListener('focus')
  onFocus() {
    this.renderer.setStyle(this.el.nativeElement,
      'background', 'yellow');
  }

  @HostListener('blur')
  onBlur() {
    this.renderer.setStyle(this.el.nativeElement,
      'background', 'white');
  }
}
```

```
ng g component custom-event-directive
```

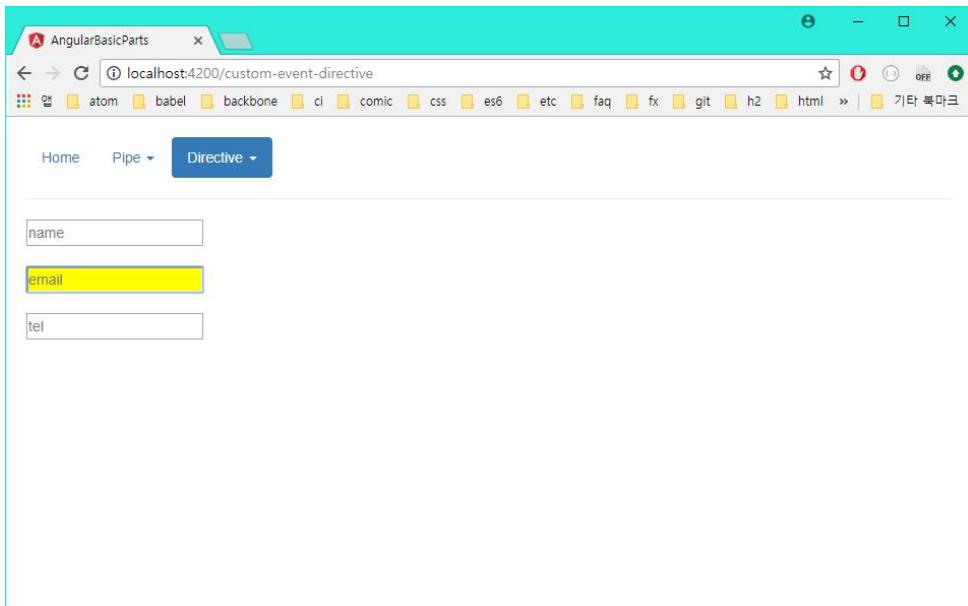
### custom-event-directive.component.html

```
<div>
  <input type="text" placeholder="name">
</div>
<br>
<div>
  <input type="email" placeholder="email" myHighlight>
</div>
<br>
<div>
  <input type="tel" placeholder="tel" myHighlight>
</div>
```

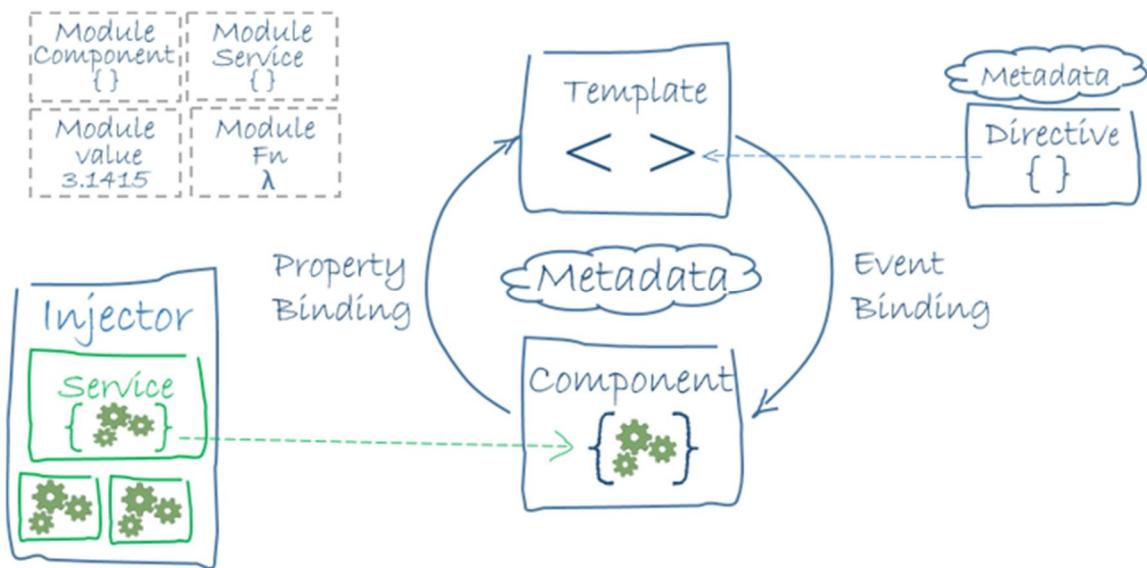
## app.component.html

```
<div class="bs-example">
<ul class="nav nav-pills">
  <li routerLinkActive="active"><a routerLink="home">Home</a></li>
  <li class="dropdown" routerLinkActive="active">
    <a href="#" data-toggle="dropdown" class="dropdown-toggle">Pipe <b class="caret"></b></a>
    <ul class="dropdown-menu">
      <li><a routerLink="builtin">Buit-in Pipe</a></li>
      <li><a routerLink="custom">Custom Pipe</a></li>
      <li><a routerLink="#">#</a></li>
      <li class="divider"></li>
      <li><a routerLink="#">#</a></li>
    </ul>
  </li>
  <li class="dropdown" routerLinkActive="active">
    <a href="#" data-toggle="dropdown" class="dropdown-toggle">Directive <b class="caret"></b></a>
    <ul class="dropdown-menu">
      <li><a routerLink="custom-directive">Custom Directive</a></li>
      <li><a routerLink="custom-directive2">Custom Directive2</a></li>
      <li class="divider"></li>
      <li><a routerLink="custom-event-directive">Custom Event Directive</a></li>
    </ul>
  </li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>
```

app-routing.module.ts 소스코드 생략



## Step 9 – Module



Angular 응용 프로그램은 모듈로 구성되며 Angular는 NgModules이라는 자체 모듈 시스템을 사용합니다. 모든 Angular 앱에는 적어도 하나의 NgModule 클래스가 있으며 루트 모듈인 AppModule이 있습니다.

루트 모듈은 작은 응용 프로그램에서 유일한 모듈일 수 있지만 대부분의 응용 프로그램은 응용 프로그램 도메인, 워크 플로 또는 밀접한 관련 기능 세트 전용 코드 모음을 포함하는 훨씬 많은 기능 모듈(feature modules)을 제공합니다.

NgModule은 @NgModule 데코레이터가 있는 클래스입니다. 데코레이터는 JavaScript 클래스를 수정하는 함수입니다. NgModule은 모듈을 설명하는 속성을 가진 단일 메타 데이터 객체를 취하는 데코레이터 함수입니다. 가장 중요한 속성은 다음과 같습니다.

- declarations - 이 모듈에 속한 뷰 클래스. Angular에는 세 가지 뷰 클래스가 있습니다. components, directives 및 pipes 입니다.
- exports - 다른 모듈의 구성 요소 템플릿에서 볼 수 있고 사용할 수 있어야 하는 선언의 하위 집합입니다.
- imports - 이 모듈에서 선언된 컴포넌트 템플릿이 필요로 하는 다른 모듈.
- providers - 글로벌 서비스 콜렉션에 기여하는 서비스 생성자; 앱의 모든 부분에서 액세스 할 수 있게 됩니다.
- bootstrap - 루트 컴포넌트라고하는 다른 모든 뷰를 호스팅하는 기본 뷰입니다.

루트 모듈 만이 부트 스트랩 속성을 설정해야 합니다.

루트 모듈은 다른 구성 요소가 루트 모듈을 가져올 필요가 없으므로 아무 것도 exports 할 필요가 없습니다.

## **루트 모듈 : AppModule**

앵귤러는 루트모듈이라는 최상위 모듈을 통해 앱을 구성한다. 앱 영역에서 사용하는 컴포넌트, 지시자, 파이프, 서비스 등과 같은 모듈을 등록하고 관리한다.

관심사에 따라 모듈을 분리하여 사용할 수 있다.

### **■ 핵심 모듈**

항상 사용하는 기능을 모은 모듈이다.

예: 타이틀 컴포넌트

### **■ 특징 모듈**

특정 기능을 처리하는 모듈이다. 관심사에 따라 모듈을 따로 구성한다. 주로 핵심 모듈에서 임포트해서 사용한다.

예: 게시판

### **■ 공유 모듈**

여러 모듈에서 반복 사용되는 기능을 별개의 모듈로 만든다. 주로 특징 모듈에서 임포트해서 사용한다.

공유 모듈은 일종의 유ти리티 모듈이다.

## 프로젝트 만들기

프로젝트가 만들어질 때 **루트 모듈**은 자동으로 생성된다. --routing=true 옵션으로 **라우팅 모듈**을 같이 만들면 편리하다.

```
ng new angular-module-example --routing=true
```

## 모듈 추가

**핵심 모듈**을 생성한다.

```
ng g module core
```

**특징 모듈**을 생성한다.

```
ng g module member
```

**특징 모듈**이면서 레이지 로딩 테스트를 위한 **모듈**을 생성한다.

```
ng g module player
```

**공유 모듈**을 생성한다.

```
ng g module share
```

## 루트 모듈 컴포넌트 생성

루트 모듈이 사용할 컴포넌트를 위한 폴더를 만든다.

```
cd src/app && mkdir component && cd ..../..
```

루트 모듈에 컴포넌트 3개를 추가한다.

```
ng g component component/intro  
ng g component component/Hello  
ng g component component/core-test
```

간편한 임포트 처리를 위한 퍼사드 파일을 만든다.

```
cd src/app/component && echo '' > index.ts && cd ..../..
```

## index.ts

```
export * from './core-test/core-test.component';
export * from './hello/hello.component';
export * from './intro/intro.component';
```

## app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';

import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';

import { CoreModule } from './core/core.module';
import { MemberModule } from './member/member.module';
import { PlayerModule } from './player/player.module';
import { ShareModule } from './share/share.module';

import { AppComponent } from './app.component';
// import { IntroComponent } from './component/intro/intro.component';
// import { HelloComponent } from './component/hello/hello.component';
// import { CoreTestComponent } from './component/core-test/core-test.component';
import { IntroComponent, HelloComponent, CoreTestComponent } from './component/index';

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    CommonModule, FormsModule, HttpClientModule,
    CoreModule.forRoot({nickName: 'Happy'}),
    MemberModule, PlayerModule,
    ShareModule
  ],
  declarations: [
    AppComponent,
    IntroComponent, HelloComponent, CoreTestComponent
  ],
  providers: []
})
```

```
bootstrap: [AppComponent]
})
export class AppModule {
  constructor(){
    console.log('AppModule # constructor() called.');
  }
}
```

#### BrowserModule

브라우저 상에서 동작한다면 반드시 포함해야 한다. 컴포넌트, 지시자, 파이프 같은 구성요소를 템플릿에 표현하는 역할을 수행한다.

#### CommonModule

ngIf, ngFor와 관련된 기능을 포함하고 있다. BrowserModule 선언 시 생략할 수 있다.

#### FormsModule

템플릿에서 자주 사용하는 NgModel 지시자나 내장 검증기 지시자 등을 포함하고 있다.

#### app-routing.module

사용자 정의 라우팅 모듈이다. 루트 모듈에 추가한 라우팅 모듈은 앱수준에서 라우팅을 수행한다.

AppRoutingModule 키워드는 클래스명이다. imports에 등록하면 라우팅등록이 완료된다.

#### declarations

앱 레벨에서 사용하고자 하는 컴포넌트, 지시자, 파이프를 선언한다.

#### bootstrap

앱 시작 컴포넌트를 등록한다.

CoreModule.forRoot({nickName: 'Happy'})

값 제공자에 매개변수로 값을 전달한다. 루트 모듈이 코어모듈을 임포트 하면서 데이터를 전달하는 방법이다.

```
import { MemberModule } from './member/member.module';
```

특정모듈을 임포트한다.

@NgModule로 선언하여 특정 범주의 기능들을 묶어서 관리한다. 모듈을 사용함으로써 각 그룹별 관리가 가능해지고 그룹내에서 공동으로 사용하는 기능을 추가한다.

exports : 다른 모듈에서 사용할 수 있도록 정의한 서브셋이다.

imports : 다른 모듈에서 exports로 선언한 서브셋을 가져온다.

@angular/common : 빌트인 파이프, 속성 디렉티브, 구조 디렉티브 ngIf, ngFor, ngSwitch 등.

@angular/core : 핵심모듈

@angular/forms : 폼 모듈

@angular/http : HTTP 모듈

@angular/platform-browser : 브라우저 모듈, 새니타이저

@angular/router : 라우터 모듈

@angular/testing : 테스트 모듈

## app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { CoreTestComponent, HelloComponent, IntroComponent } from './component/index';

const routes: Routes = [
  { path: '', redirectTo: '/intro', pathMatch: 'full' },
  { path: 'intro', component: IntroComponent },
  { path: 'hello', component: HelloComponent },
  { path: 'core-test', component: CoreTestComponent },
  { path: 'lazy', loadChildren: 'app/player/player.module#PlayerModule' },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
```

```
export class AppRoutingModule { }
```

router-outlet 위치에 교대로 표시할 컴포넌트를 라우팅 모듈 설정에 등록한다.

#### routes

라우팅 설정을 담고 있다. path에 설정된 URL 요청을 지정된 컴포넌트로 라우팅되게 한다.

loadChildren: '모듈파일#모듈클래스'

페이지 모듈 로딩이다. 사용자가 요청하는 시점에 모듈을 비동기적으로 로딩한다.

```
{ path: 'lazy', loadChildren: 'app/player/player.module#PlayerModule' }
```

http://localhost:4200/lazy 요청이 들어오면

src/app/player/player-routing.module.ts에 선언된 내용에 따라 처리한다.

```
RouterModule.forChild([
  { path: 'player', component: PlayerComponent }
])
```

URL Pattern이 '/player'인 경우 PlayerComponent를 실행하여 기동 컴포넌트의 라우터-아울렛에 표시된다.

#### index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>AngularModuleExample</title>
    <base href="/">

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
```

```

<style type="text/css">
    .bs-example {
        margin: 20px;
    }
</style>

</head>
<body>
    <app-root></app-root>
</body>
</html>

```

## app.component.html

```

<div class="bs-example">
    <ul class="nav nav-pills">
        <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
            <a [routerLink]="/, 'intro']">
                Root Module <b class="glyphicon glyphicon-chevron-right"></b> Intro</a>
            </li>
        <li class="dropdown" routerLinkActive="active">
            <a href="#" data-toggle="dropdown" class="dropdown-toggle">
                Root Module <b class="caret"></b></a>
            <ul class="dropdown-menu">
                <li><a routerLink="hello">Hello</a></li>
                <li class="divider"></li>
                <li><a routerLink="#">#</a></li>
            </ul>
        </li>
        <li class="dropdown" routerLinkActive="active">
            <a href="#" data-toggle="dropdown" class="dropdown-toggle">
                Core Module <b class="caret"></b></a>
            <ul class="dropdown-menu">
                <!-- CoreModule 의 TitleComponent 를 루트모듈의 CoreTestComponent 에서 사용합니다. -->
                <li><a routerLink="core-test">Core Test</a></li>
                <li class="divider"></li>
                <li><a routerLink="#">#</a></li>
            </ul>
        </li>
        <li class="dropdown" routerLinkActive="active">
            <a href="#" data-toggle="dropdown" class="dropdown-toggle">

```

```
Feature Module <b class="caret"></b> </a>
<ul class="dropdown-menu">
  <!-- MemberModule 의 MemberListComponent 를 사용합니다. -->
  <li><a routerLink="member-list">Member List</a></li>
  <li class="divider"></li>
  <!-- PlayerModule 을 정적으로 연동한다. -->
  <li><a routerLink="player">Player</a></li>
  <!-- PlayerModule 을 동적으로 연동한다. 실제 사용할 때 객체화 하는 레이저로딩이다. -->
  <li><a routerLink="lazy/player">Lazy Player</a></li>
</ul>
</li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>
```

## 핵심 모듈 : CoreModule

모든 모듈에서 사용해야 하는 구성요소를 묶은 모듈이다.

핵심 모듈 → 루트 모듈

```
ng g component core/title --flat
```

### title.component.ts

```
import { Component, OnInit, Input } from '@angular/core';
import { UserService } from './user.service';

@Component({
  selector: 'app-title',
  templateUrl: './title.component.html',
  styleUrls: ['./title.component.css']
})
export class TitleComponent implements OnInit {

  @Input() title = '';
  user = '';

  constructor(userService: UserService) {
    this.user = userService.nickName;
  }

  ngOnInit() {
  }

}
```

### title.component.html

```
<h2 highlight>{{title}}</h2> by <b>{{user}}</b>
```

highlight 디렉티브가 적용되려면 컴포넌트를 갖고 있는 CoreModule 모듈안에 해당 디렉티브가 있거나, 디렉티브를 제공하는 모듈을 CoreModule이 직접 임포트해야 한다.

```
<app-title [title] ="타이틀"></app-title>
```

타이틀 컴포넌트를 다른 컴포넌트가 템플릿에서 위처럼 설정하면 title 속성(키 역할)을 통해 문자열을 받아 사용할 수 있다.

```
ng g service core/user
```

### user.service.ts

```
import { Injectable, Optional } from '@angular/core';

export class UserServiceConfig {
  nickName = "";
}

@Injectable()
export class UserService {
  private _nickName = "";

  constructor(@Optional() config: UserServiceConfig) {
    if (config) {
      this._nickName = config.nickName;
    }
  }

  get nickName() {
    return this._nickName;
  }
}
```

#### @Optional

주입 객체가 있다면 사용하고 아니면 null을 주입한다. null 체크 로직이 부가적으로 필요하다. 주입대상 객체가 없어서 발생하는 예외를 막는다.

#### get nickName()

게터메소드로 정의하면 서비스객체.nickName 과 같이 접근하여 값을 얻어갈 수 있다.

get/set 으로 선언된 함수는 UserServiceConfig 가 함수로써 갖고 있는 prototype 프로퍼티가 가리키는 상속용 객체에 접근자 프로퍼티로 설정되어 존재하게 된다.

## core.module.ts

```
import { NgModule, Optional, SkipSelf, ModuleWithProviders } from '@angular/core';
import { CommonModule } from '@angular/common';

import { TitleComponent } from './title.component';

import { UserService, UserServiceConfig } from './user.service';

@NgModule({
  imports: [
    CommonModule
  ],
  exports: [TitleComponent], // 서비스는 명시적 익스포트 설정 대상이 아니다. 자동으로 제공된다.
  declarations: [TitleComponent],
  providers: [UserService]
})
export class CoreModule {

  // 부모 주입기에 CoreModule 이 존재하는지 체크하기 위해서 @SkipSelf 를 사용합니다.
  // @Optional 데코레이터는 객체가 있으면 전달하고 없으면 null 을 전달하게 합니다.
  constructor(@SkipSelf() @Optional() parentModule: CoreModule) {
    console.log('CoreModule # constructor() called.');
    if (parentModule) {
      // 코어 모듈은 전체 모듈이 공유해서 언제나 사용하는 기능을 가진 모듈입니다.
      // 코어 모듈은 하나만 존재하면 됩니다. 코어 모듈은 루트 모듈이 임포트해서 사용하는 방식입니다.
      throw new Error('CoreModule 이 이미 로딩되었습니다.');
    }
  }

  static forRoot(config: UserServiceConfig): ModuleWithProviders {
    return {
      ngModule: CoreModule,
      providers: [
        { provide: UserServiceConfig, useValue: config }
      ]
    };
  }
}

exports: [TitleComponent]
```

루트모듈에서 핵심모듈의 컴포넌트를 사용하기 위해서 외부로 노출한다. 서비스, 일반 함수, 모델 클래스는 exports 설정을 하지 않는다.

```
constructor( @Optional() @SkipSelf() parentModule: CoreModule)
```

생성자에서 핵심모듈 CoreModule을 주입받고자 시도한다. 부모 주입기에 핵심모듈이 이미 생성됐는지 검사한다.

```
@SkipSelf()
```

주입기 체계에서 자신은 건너 뛰고 자신으로부터 위에 존재하는 부모 주입기에서 CoreModule 을 찾는다.

```
static forRoot(config: UserServiceConfig): ModuleWithProviders
```

루트 모듈에서 CoreModule.forRoot({nickName: 'Happy'}) 설정으로 객체를 매개변수로 넘기면 받아서 사용한다.

루트 모듈이 코어모듈의 작동방식을 제어하기 위한 환경설정의 정보를 전달하는 용도로 사용될 수 있다.

## 특징 모듈 : PlayerModule

player 모듈이 레이지 로딩할 때 기동 컴포넌트를 담당할 컴포넌트를 만든다.

```
ng g component player/player --flat
```

### player.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { Routes, RouterModule } from '@angular/router';
import { PlayerRoutingModule } from './player-routing.module';

import { ShareModule } from '../share/share.module';

import { PlayerComponent } from './player.component';

@NgModule({
  imports: [
    CommonModule,
    PlayerRoutingModule,
    ShareModule
  ],
  declarations: [PlayerComponent]
})
export class PlayerModule {
  constructor() {
    console.log('PlayerModule # constructor() called.');
    console.log(new Date().toLocaleString());
  }
}
```

### player.component.html

```
<div highlight>{{ "PlayeR" | myUpper }}</div>
```

#### highlight

해당 디렉티브는 모듈에서 임포트한 SharedModule에서 제공한다.

## myUpper

해당 파일은 모듈에서 임포트한 SharedModule에서 제공한다.

```
cd src/app/player && echo '' > player-routing.module.ts && cd ../../..
```

## player-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule } from '@angular/router';

import { PlayerComponent } from './player.component';

@NgModule({
  imports: [
    RouterModule.forChild([
      { path: 'player', component: PlayerComponent }
    ])
  ],
  exports: [RouterModule]
})
export class PlayerRoutingModule { }
```

앱 전체에서 forRoot 함수는 한 번만 사용해야 합니다. 기동 모듈이 아닌 모듈에 라우팅 정보를 설정할 때 forChild 함수를 사용합니다.

## 특징 모듈 : MemberModule

만약 루트모듈에 모든 모듈을 구성하려고 한다면 모듈 구성이 복잡해질 것이다. 따라서 루트 모듈에서 하위 모듈로 분리할 필요가 있다. 이렇게 하위로 분리하는 모듈을 특징 모듈이라 부른다. 특징 모듈은 관심사가 같은 모듈을 묶어 상위 모듈에게 제공한다. 라우팅 모듈은 일반적으로 분리해서 특징모듈로써 사용한다.

특징 모듈 라우팅 모듈 → 특징 모듈 → **루트 모듈** ← 라우팅 모듈(앱 레벨)

ng g directive member/highlight

### highlight.directive.ts

```
import { Directive, ElementRef, HostListener, Input } from '@angular/core';

@Directive({
  selector: '[highlight]'
})
export class HighlightDirective {
  private el: HTMLElement;

  constructor(el: ElementRef) {
    this.el = el.nativeElement;
    this.el.style.fontSize = "30px";
  }

  @HostListener('mousemove')
  onMouseMove() {
    this.el.style.backgroundColor = "blue";
    this.el.style.color = "white";
  }

  @HostListener('mouseleave')
  onMouseLeave() {
    this.el.style.backgroundColor = null;
    this.el.style.color = "black";
  }
}
```

@HostListener

디렉티브가 사용되는 엘리먼트에서 발생하는 이벤트를 듣는다. 등록된 이벤트에 따라 지시자 선언 엘리먼트 영역에 마우스가 위치하면 배경색을 blue로 바꾼다.

```
ng g service member/member
```

### member.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { MemberRoutingModule } from './member-routing.module';

import { HighlightDirective } from './highlight.directive';
import { MemberListComponent } from './member-list.component';

import { MemberService } from './member.service';

// MemberModule에 이미 HighlightDirective가 존재하는데
// SharedModule에도 같은 이름의 HighlightDirective가 존재한다.
// 이 경우, 모듈 자신의 디렉티브가 사용된다.
import { SharedModule } from '../share/share.module';

@NgModule({
  imports: [
    CommonModule,
    FormsModule,
    MemberRoutingModule,
    SharedModule
  ],
  declarations: [HighlightDirective, MemberListComponent],
  providers: [MemberService]
})
export class MemberModule {
  constructor(){
    console.log('MemberModule # constructor() called.');
  }
}
```

특징 모듈은 루트 모듈의 하위로 관심사를 별도로 갖고 있다. 특징 모듈은 bootstrap 속성을 사용하지 않는다.

```
cd src/app/member && echo '' > member-routing.module.ts && cd ../../..
```

### member.service.ts

```
import { Injectable } from '@angular/core';

export class Member {
  constructor(
    public name: string,
    public age: number) { }

}

const MEMBERS: Member[] = [
  new Member('유비', 30),
  new Member('관우', 29),
  new Member('장비', 28)
];

@Injectable()
export class MemberService {

  getMembers() {
    // 프라미스 객체를 리턴한다. 배열은 1초 후 비동기적으로 전달된다.
    return new Promise<Member[]>(resolve => {
      setTimeout(() => { resolve(MEMBERS); }, 1000);
    });
  }

  getMember(name: string) {
    return this.getMembers()
      .then(members => members.find(member => member.name === name));
  }

}
```

```
ng g component member/member-list --flat
```

### **member-list.component.ts**

```
import { Component, OnInit } from '@angular/core';
import { Member, MemberService } from './member.service';

@Component({
  selector: 'app-member-list',
  templateUrl: './member-list.component.html',
  styleUrls: ['./member-list.component.css']
})
export class MemberListComponent implements OnInit {
  members: Member[];
  age: number;
  name: string;

  constructor(private memberService: MemberService) { }

  ngOnInit() {
    this.memberService.getMembers().then(members => {
      this.members = members;
    });
    this.setAge("유비");
  }

  setAge(name: string) {
    this.name = name;
    this.memberService.getMember(name).then(member => {
      this.age = member.age;
    });
  }
}
```

### **member-list.component.html**

```
 {{name}}의 나이 : {{age}}
<br>
<div *ngFor='let m of members' highlight (mouseover)="setAge(m.name)">
  {{m.name}} {{m.age}}
</div>
```

멤버 리스트 멤버의 div 영역에 마우스가 위치하면 setAge 함수가 호출되고, 이름으로 MemberService로부터 해당하는 멤버 정보를 1초 후에 얻어온다.

### member-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule } from '@angular/router';

import { MemberListComponent } from './member-list.component';

@NgModule({
  imports: [
    RouterModule.forChild([
      { path: 'member-list', component: MemberListComponent }
    ])
  ],
  exports: [RouterModule]
})
export class MemberRoutingModule { }
```

RouterModule.forChild

루트 모듈의 라우팅 모듈에 설정하지 않고 특징 모듈의 라우팅 모듈에서 특징 모듈과 관련된 라우팅 설정을 추가한다.

## 공유 모듈 : SharedModule

자주 반복적으로 사용되는 모듈이 공유 모듈 대상이다. 루트 모듈은 특징 모듈을 임포트하고 특징 모듈은 공유 모듈을 임포트한다. BrowserModule, FormsModule은 일반적으로 공유 모듈로 묶고 특징 모듈에 제공해서 사용한다.

공유 모듈 → 특징 모듈 → **루트 모듈**

### highlight.directive.ts

```
import { Directive, ElementRef, HostListener, Input } from '@angular/core';

@Directive({
  selector: '[highlight]'
})
export class HighlightDirective {
  private el: HTMLElement;

  constructor(el: ElementRef) {
    this.el = el.nativeElement;
    this.el.style.fontSize = "30px";
  }

  @HostListener('mousemove')
  onMouseMove() {
    this.el.style.backgroundColor = "red";
    this.el.style.color = "white";
  }

  @HostListener('mouseleave')
  onMouseLeave() {
    this.el.style.backgroundColor = null;
    this.el.style.color = "black";
  }
}
```

```
ng g pipe share/my-upper
```

### my-upper.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'myUpper'
})
export class MyUpperPipe implements PipeTransform {

  transform(value: string): any {
    return value.toUpperCase();
  }
}
```

### share.module.ts

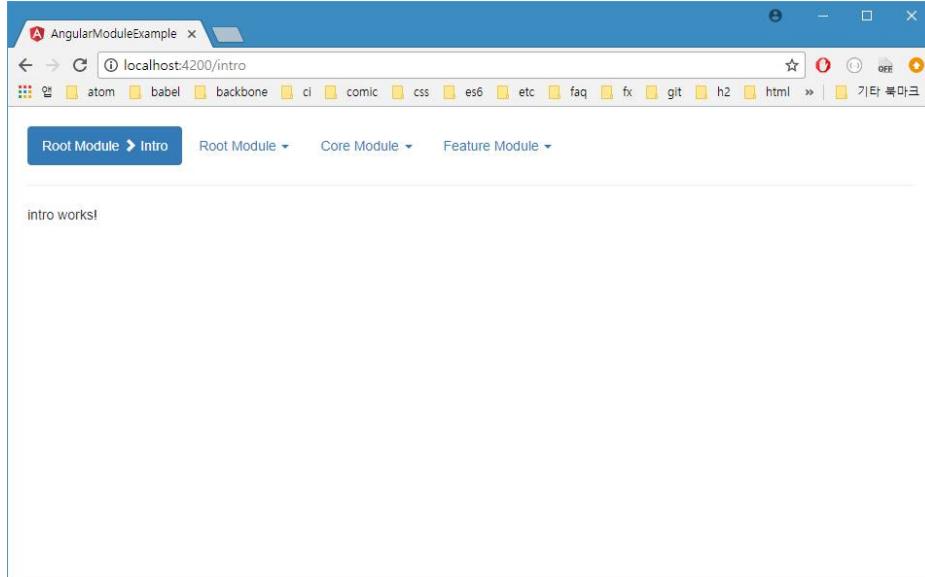
```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';

import { MyUpperPipe } from './my-upper.pipe';
import { HighlightDirective } from './highlight.directive';

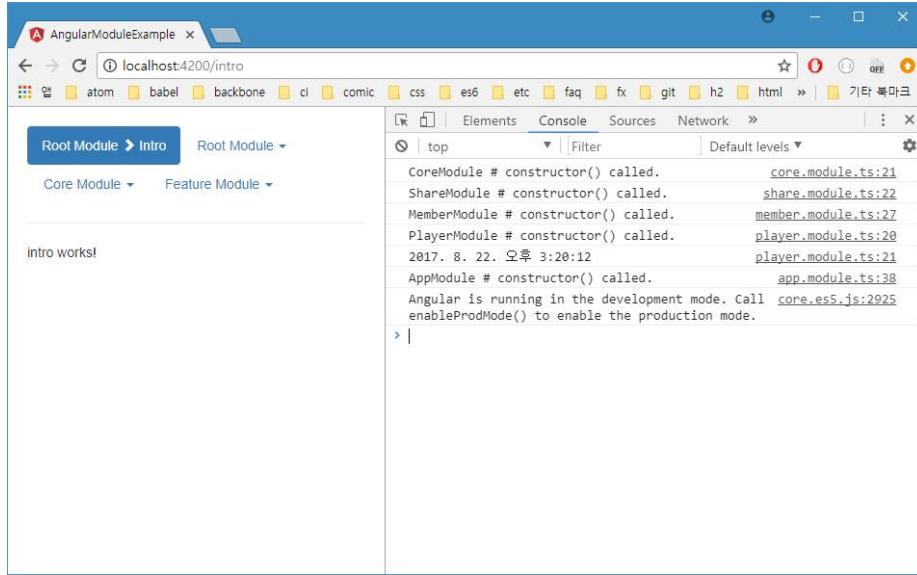
@NgModule({
  imports: [
    CommonModule
  ],
  exports: [
    MyUpperPipe,
    HighlightDirective,
    CommonModule,
    FormsModule
  ],
  declarations: [MyUpperPipe, HighlightDirective]
})
export class ShareModule {
  constructor(){
    console.log('ShareModule # constructor() called.');
  }
}
```

```
}
```

```
}
```

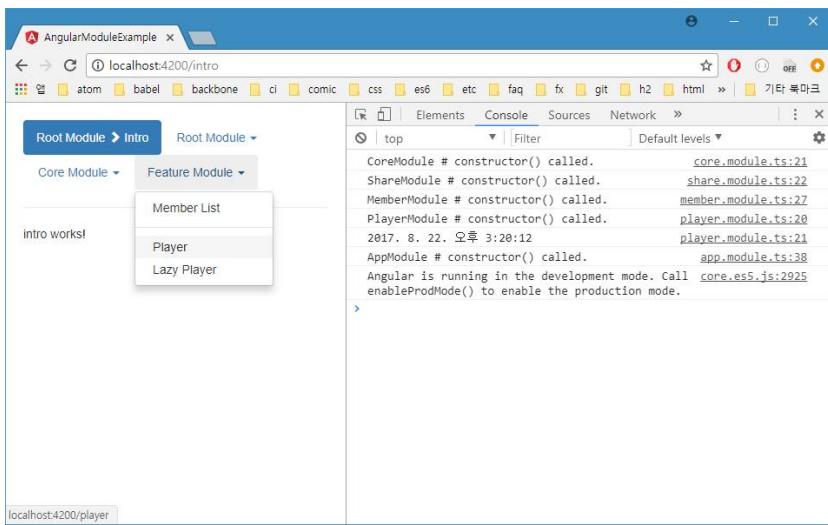


F12 키를 눌러서 개발자모드의 콘솔을 확인해 보자.

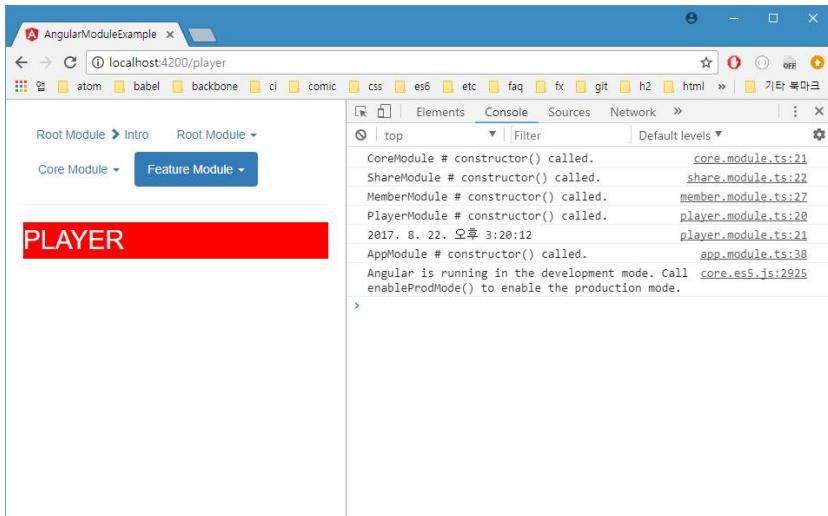


핵심모듈, 공유모듈, 특징모듈이 만들어지고 루트모듈이 생성됨을 알 수 있다.

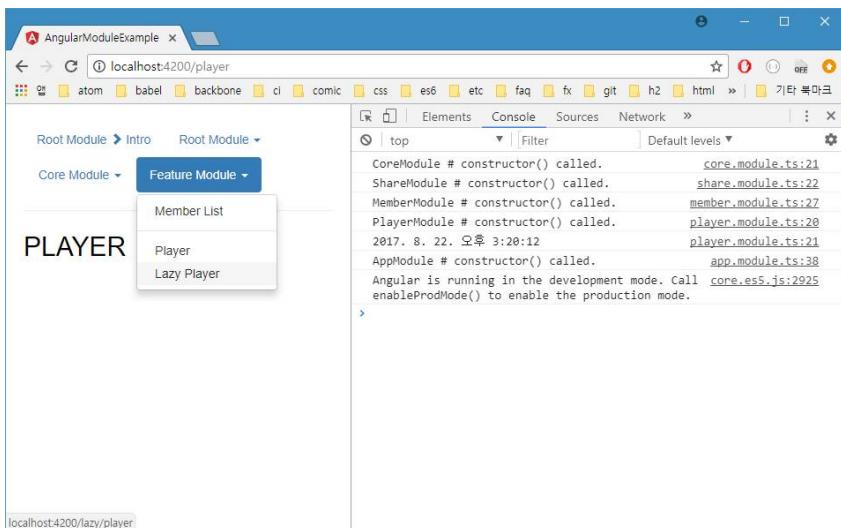
```
CoreModule # constructor() called.
ShareModule # constructor() called.
MemberModule # constructor() called.
PlayerModule # constructor() called.
2017. 8. 22. 오후 3:20:12
AppModule # constructor() called.
```



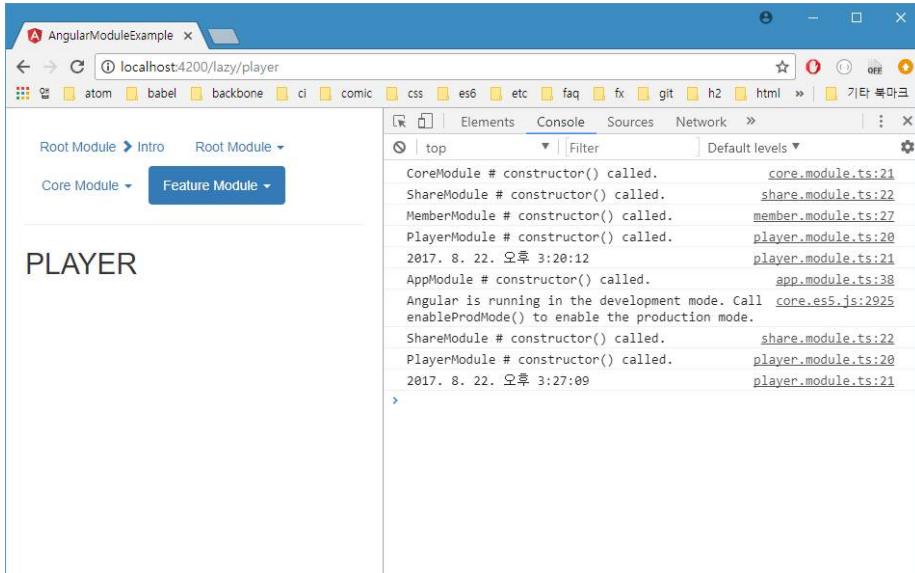
특징모듈 PlayerModule의 PlayerComponent를 연동한다. PlayerModule 모듈은 이미 객체이므로 로그에 변화는 없다.



PlayerComponent의 화면에는 공유모듈 ShareModule의 HighlightDirective 와 MyUpperPipe 가 적용됨을 알 수 있다.

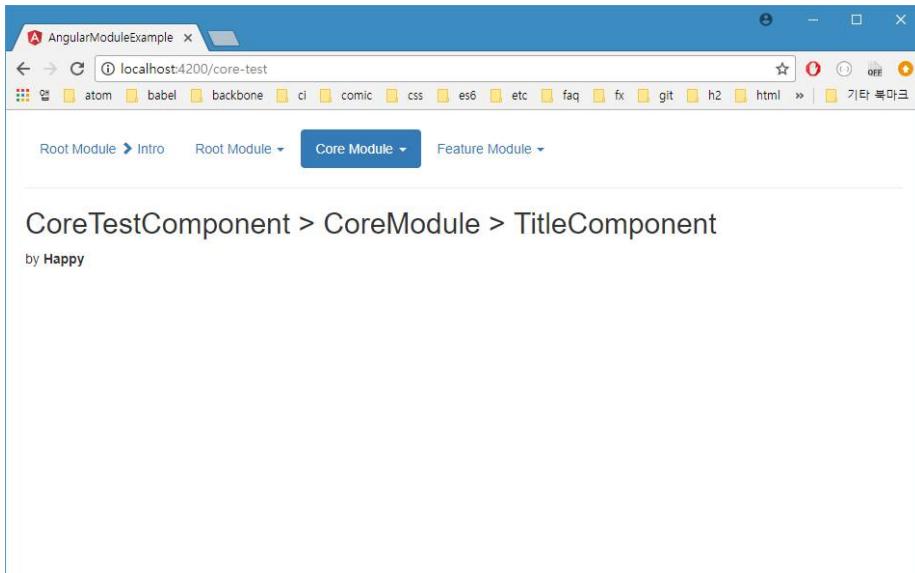


이번에는 Lazy Player 링크를 클릭해보자.



레이저로딩 설정이 된 모듈은 실제로 해당 모듈을 사용하고자 할 때 객체가 된다. PlayerModule이 사용하는 SharedModule도 이 때, 같이 객체가 생성되었음을 알 수 있다.

```
 SharedModule # constructor() called.
PlayerModule # constructor() called.
```



루트 모듈의 CoreTestComponent 가 핵심 모듈의 TitleComponent를 사용하는 예제이다.

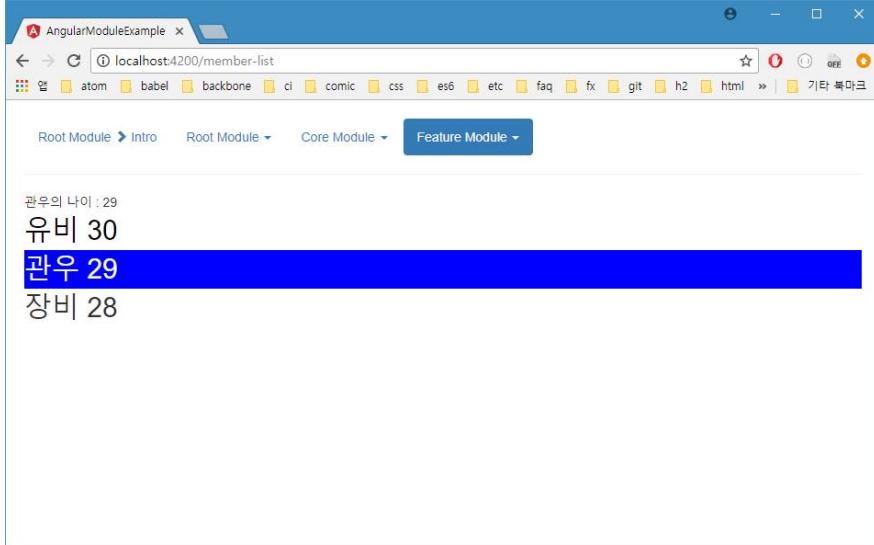
## core-test.component.html

```
<app-title [title]="title"></app-title>
```

## app.component.html

```
<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a [routerLink]=["/", 'intro']>
        Root Module <b class="glyphicon glyphicon-chevron-right"></b> Intro</a>
    </li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">
        Root Module <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="hello">Hello</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
      </ul>
    </li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">
        Core Module <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <!-- CoreModule 의 TitleComponent 를 루트모듈의 CoreTestComponent 에서 사용합니다. -->
        <li><a routerLink="core-test">Core Test</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
      </ul>
    </li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">
        Feature Module <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <!-- MemberModule 의 MemberListComponent 를 사용합니다. -->
        <li><a routerLink="member-list">Member List</a></li>
        <li class="divider"></li>
        <!-- PlayerModule 을 정적으로 연동한다. -->
        <li><a routerLink="player">Player</a></li>
        <!-- PlayerModule 을 동적으로 연동한다. 실제 사용할 때 객체화 하는 레이저로딩이다. -->
        <li><a routerLink="lazy/player">Lazy Player</a></li>
      </ul>
    </li>
  </ul>
</div>
```

```
</ul>
</li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>
```



특정 모듈 MemberModule의 MemberListComponent를 사용하는 모습이다.

member-routing.module.ts

```
RouterModule.forChild([
  { path: 'member-list', component: MemberListComponent }
])
```



## Chapter 3 : Angular Core

컴포넌트와 디렉티브의 라이프 사이클을 살펴봅니다.

컴포넌트들 사이에 대화방법을 살펴봅니다.

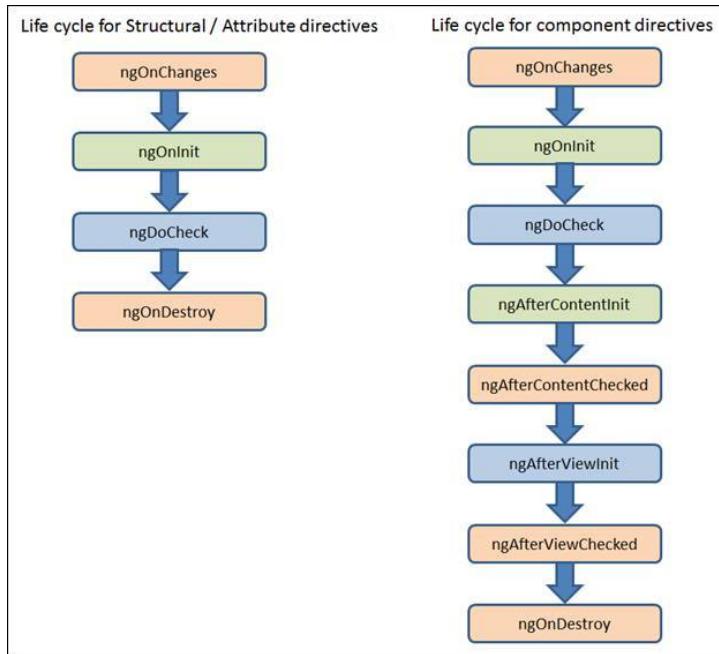
특정 상황에서 처리해야 하는 로직을 어떻게 설정해서 사용하는지 학습합니다.

사용자가 입력한 데이터의 유효성을 체크하고 데이터에 에러가 있는 경우, 어떻게 사용자에게 통보하는지 살펴봅니다.

- Life Cycle
- Component Communication
- HTTP
- Router
- Guard
- Form

# Step 1 – Life Cycle

컴포넌트와 디렉티브는 생명주기를 갖는다. 디렉티브는 화면이 없기 때문에 화면처리와 관련된 메소드들은 존재하지 않는다. ngAfterContentInit, ngAfterContentChecked, ngAfterViewInit, ngAfterViewChecked 메소드들은 화면과 관련된 흑 메소드들이다.



status 흑 메소드들은 변경감지와 관련되어 있다.

	event	init	status	설명
ngOnChanges	v			속성 바인딩의 상태 값이 변경될 때 호출. [key] = "var"
<b>ngOnInit</b>		v		컴포넌트 클래스 생성자가 호출되어 객체가 만들어 진 후, 초기화 작업 용도
ngDoCheck			v	<b>엘리먼트의 상태가 변할 때마다 호출</b>
ngAfterContentInit		v		컨텐츠 초기화 : 컨텐츠가 뷰에 장착된 후
ngAfterContentChecked			v	프로젝트의 외부 컨텐츠와 바인딩을 점검한 후 호출
ngAfterViewInit		v		뷰 초기화 : 뷰가 화면에 표시된 후
ngAfterViewChecked			v	컴포넌트나 자식뷰의 바인딩을 점검한 후 호출
<b>ngOnDestroy</b>	v			컴포넌트나 디렉티브가 파괴되기 전

```
ng new angular-life-cycle
```

### app.component.html

아무것도 화면에 표시하지 않고 로그로만 테스트 해 봅시다.

```
// nothing
```

### app.component.ts

```
import { Component,
  OnChanges, OnInit, DoCheck,
  AfterContentInit, AfterContentChecked, AfterViewInit,
  AfterViewChecked, OnDestroy } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnChanges, OnInit, DoCheck, AfterContentInit, AfterContentChecked,
AfterViewInit, AfterViewChecked, OnDestroy {

  constructor() {
    console.log("0. AppComponent # constructor() called.");
  }

  ngOnChanges(changes) {
    console.log("1. event 1 : ngOnChanges");
  }

  ngOnInit() {
    console.log("2. init 1 : ngOnInit");
  }

  ngDoCheck() {
    console.log("3. status 1 : ngDoCheck");
  }

  ngAfterContentInit() {
    console.log("4. init 2 : ngAfterContentInit");
  }
}
```

```

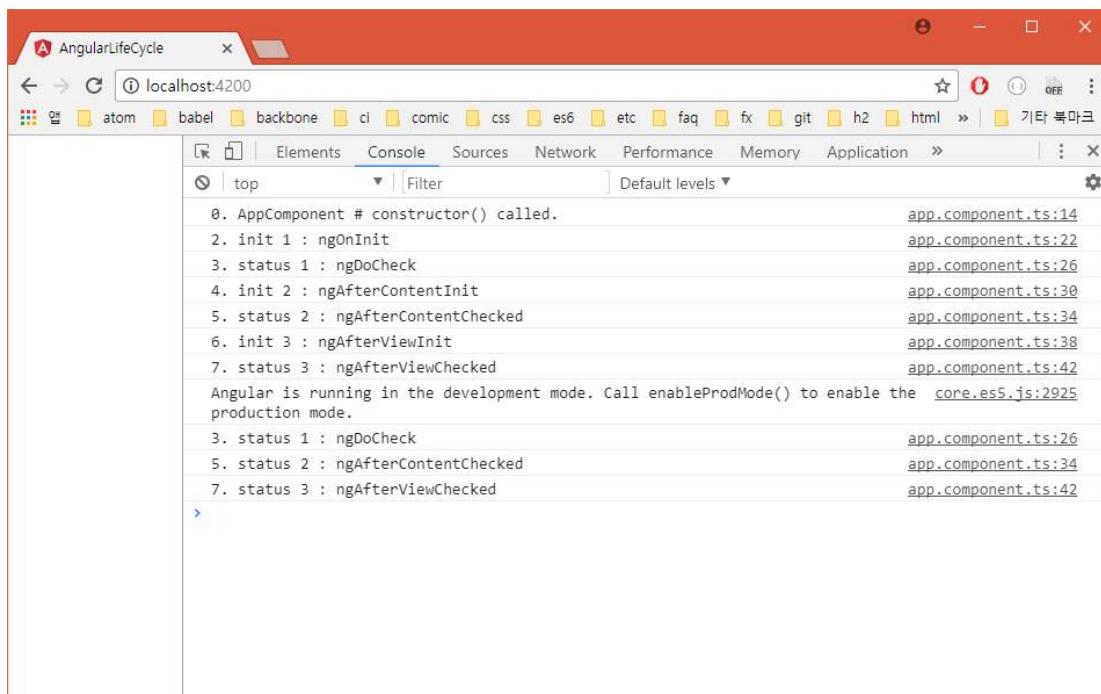
ngAfterContentChecked() {
  console.log("5. status 2 : ngAfterContentChecked");
}

ngAfterViewInit() {
  console.log("6. init 3 : ngAfterViewInit");
}

ngAfterViewChecked() {
  console.log("7. status 3 : ngAfterViewChecked");
}

ngOnDestroy() {
  console.log("8. event 2 : ngOnDestroy");
}
}

```



```

0. AppComponent # constructor() called.
1. init 1 : ngOnInit
2. status 1 : ngDoCheck
3. init 2 : ngAfterContentInit
4. status 2 : ngAfterContentChecked

```

```
6. init 3 : ngAfterViewInit
7. status 3 : ngAfterViewChecked
core.es5.js:2925 Angular is running in the development mode. Call enableProdMode() to enable the production mode.
3. status 1 : ngDoCheck
5. status 2 : ngAfterContentChecked
7. status 3 : ngAfterViewChecked
```

컴포넌트 클래스가 객체가 되기 위해서 생성자가 제일 먼저 호출됩니다.

AppComponent는 기동 컴포넌트이므로 외부에서 값을 전달 받지 못합니다. 따라서, ngOnChanges 함수는 작동하지 않습니다. 이 메소드는 조건 기동 함수입니다.

```
ngOnChanges(changes) {
  console.log("1. event 1 : ngOnChanges");
}
```

라이프 사이클 흐름 메소드는 매번 작동순서대로 기동합니다.

변경감지가 작동하면서 status 흐름 메소드(ngDoCheck, ngAfterContentChecked, ngAfterViewChecked)가 호출됐다는 것을 알 수 있습니다.

### app.component.ts

클래스 내 메소드들을 주석처리 합니다.

```
import { Component,
  OnChanges, OnInit, DoCheck,
  AfterContentInit, AfterContentChecked, AfterViewInit,
  AfterViewChecked, OnDestroy } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent
  // implements OnChanges, OnInit, DoCheck, AfterContentInit,
  // AfterContentChecked, AfterViewInit, AfterViewChecked, OnDestroy
  {

  // constructor() {
```

```

//   console.log("0. AppComponent # constructor() called.");
// }
//
// ngOnChanges(changes) {
//   console.log("1. event 1 : ngOnChanges");
// }
//
// ngOnInit() {
//   console.log("2. init 1 : ngOnInit");
// }
//
// ngDoCheck() {
//   console.log("3. status 1 : ngDoCheck");
// }
//
// ngAfterContentInit() {
//   console.log("4. init 2 : ngAfterContentInit");
// }
//
// ngAfterContentChecked() {
//   console.log("5. status 2 : ngAfterContentChecked");
// }
//
// ngAfterViewInit() {
//   console.log("6. init 3 : ngAfterViewInit");
// }
//
// ngAfterViewChecked() {
//   console.log("7. status 3 : ngAfterViewChecked");
// }
//
// ngOnDestroy() {
//   console.log("8. event 2 : ngOnDestroy");
// }

}

```

### app.component.html

```
<app-parent></app-parent>
```

```
ng g component parent --flat
```

### **parent.component.ts**

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-parent',
  templateUrl: './parent.component.html',
  styleUrls: ['./parent.component.css']
})
export class ParentComponent implements OnInit {
  title = 'ParentComponent';
  shouldShow = true;

  constructor() {}

  ngOnInit() {}

  toggle() {
    this.shouldShow = !this.shouldShow;
  }
}
```

### **parent.component.html**

```
<h1>{{title}}</h1>
<button (click)="toggle()">shouldShow toggle</button>
<br /><br />
<fieldset>
  <app-child> </app-child>
  <!-- <app-child [prop]="shouldShow" [content]=""외부로부터 전달받는 값""></app-child> -->
</fieldset>
```

```
ng g component child --flat
```

### **child.component.ts**

```
import { Component, Input,
  OnChanges, OnInit, DoCheck,
  AfterContentInit, AfterContentChecked, AfterViewInit,
```

```

AfterViewChecked, OnDestroy } from '@angular/core';

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent implements OnChanges, OnInit, DoCheck, AfterContentInit, AfterContentChecked, AfterViewInit, AfterViewChecked, OnDestroy {
  title = 'ChildComponent';
  message = "";

  constructor() {
    // 객체 생성 시점에 호출된다.
    // 부모 컴포넌트가 속성을 설정하여 전달하는 파라미터를 여기서는 이용할 수 없다.
    // 대신 라이프사이클 함수중에 하나를 이용하자.
    console.log("0. ChildComponent # constructor() called.");
    console.log("this.content = " + this.content); // undefined
  }

  @Input() content;
  @Input()
  set prop(name: string) {
    console.log("@Input prop() : name = " + name);
    console.log("@Input content : this.content = " + this.content);
  }

  ngOnChanges(changes) {
    // 외부로부터 상태를 전달받을 때에만 매번 호출된다.
    console.log("1. event 1 : ngOnChanges");
    console.log("this.content = " + this.content); // 출력
    console.log(changes); // {content: SimpleChange, prop: SimpleChange}
    console.log(JSON.stringify(changes));
  }

  ngOnInit() {
    // 사용한 자식 컴포넌트와 �렉티브가 초기화되기 전 호출된다.
    console.log("2. init 1 : ngOnInit");
  }

  ngDoCheck() {
    // 상태가 변할 때마다 직전에 호출된다.
    console.log("3. status 1 : ngDoCheck");
  }
}

```

```

}

ngAfterContentInit() {
    // 자식 컴포넌트와 디렉티브가 초기화된 후 호출된다.
    console.log("4. init 2 : ngAfterContentInit");
}

ngAfterContentChecked() {
    // 상태가 변할 때마다 바인딩이 체크된 후에 호출된다.
    console.log("5. status 2 : ngAfterContentChecked");
}

ngAfterViewInit() {
    // 자신의 뷰와 자식의 뷰가 모두 초기화된 후 호출된다.
    console.log("6. init 3 : ngAfterViewInit");
}

ngAfterViewChecked() {
    // 상태가 변할 때마다 콘텐츠가 표시된 후 호출된다.
    console.log("7. status 3 : ngAfterViewChecked");
}

ngOnDestroy() {
    // 컴포넌트나 디렉티브가 제거되기 전에 호출된다.
    // 리소스 반환, 컴포넌트 소멸 전에 수행해야 하는 로직을 주로 배치한다.
    console.log("8. event 2 : ngOnDestroy");
}

}

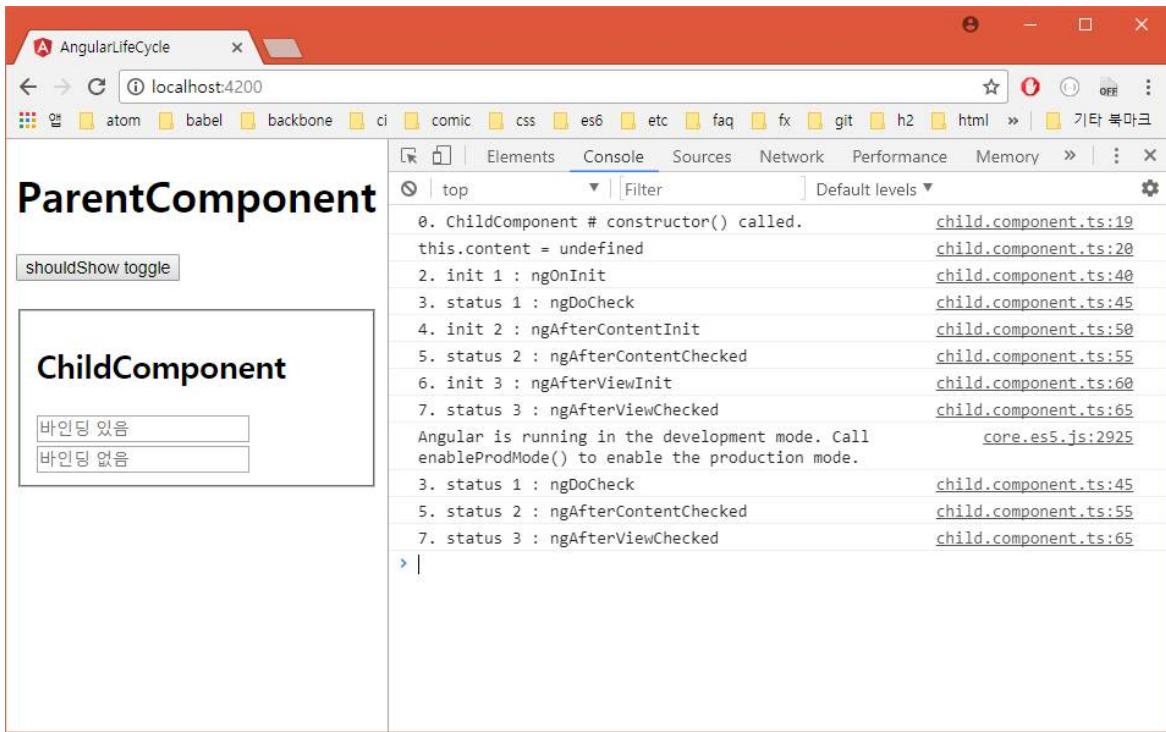
```

### **child.component.html**

```

<h2>{{title}}</h2>
<input type="text" [(ngModel)]="message" placeholder="바인딩 있음">
<input type="text" placeholder="바인딩 없음">

```



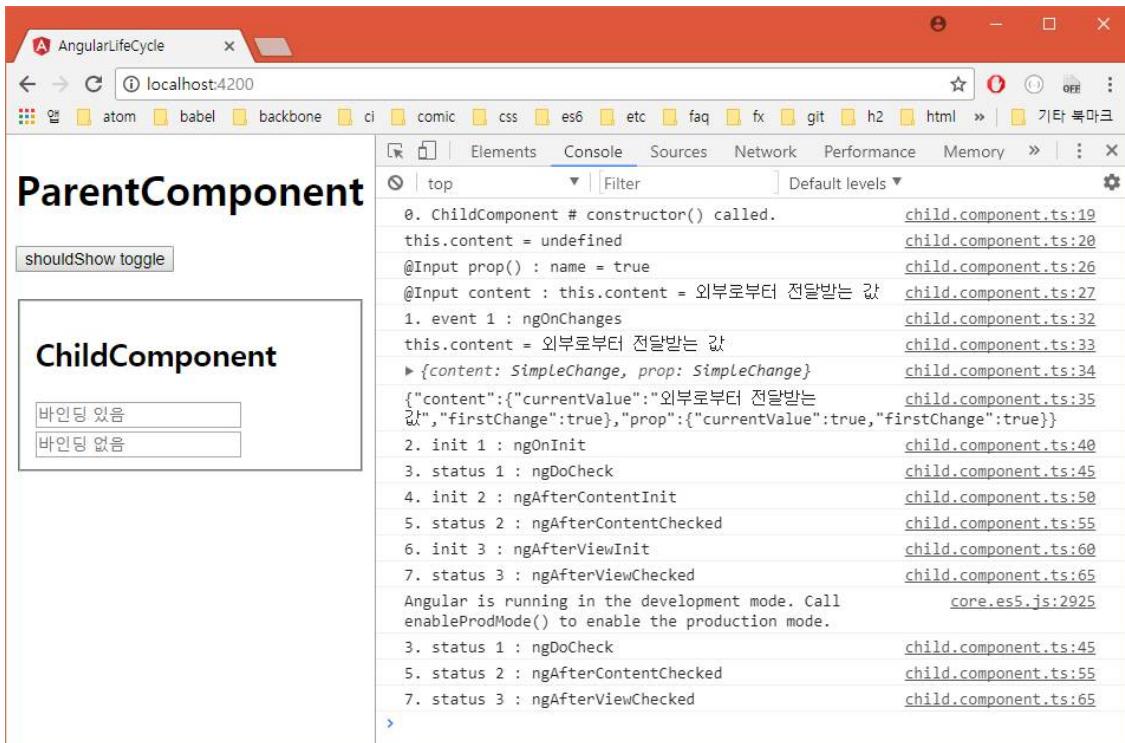
다음처럼 코드를 변경하고 다시 결과를 확인해 보자.

### parent.component.html

```
<h1>{{title}}</h1>
<button (click)="toggle()">shouldShow toggle</button>
<br /><br />
<fieldset>
  <!-- <app-child> </app-child> -->
  <app-child [prop]="shouldShow" [content]=""외부로부터 전달받는 값""></app-child>
</fieldset>
```

prop 키로 컴포넌트의 변수 shouldShow의 값을 전달한다. content 키로 문자열을 전달한다.

이 때, 자식 컴포넌트의 ngOnChanges 메소드가 기동한다.



0. ChildComponent # constructor() called.

this.content = undefined

@Input prop() : name = true

@Input content : this.content = 외부로부터 전달받는 값

1. event 1 : ngOnChanges

this.content = 외부로부터 전달받는 값

{content: SimpleChange, prop: SimpleChange}

{"content":{"currentValue":"외부로부터 전달받는 값","firstChange":true}, "prop":{"currentValue":true,"firstChange":true}}

2. init 1 : ngOnInit

3. status 1 : ngDoCheck

4. init 2 : ngAfterContentInit

5. status 2 : ngAfterContentChecked

6. init 3 : ngAfterViewInit

7. status 3 : ngAfterViewChecked

core.es5.js:2925 Angular is running in the development mode. Call enableProdMode() to enable the production mode.

3. status 1 : ngDoCheck

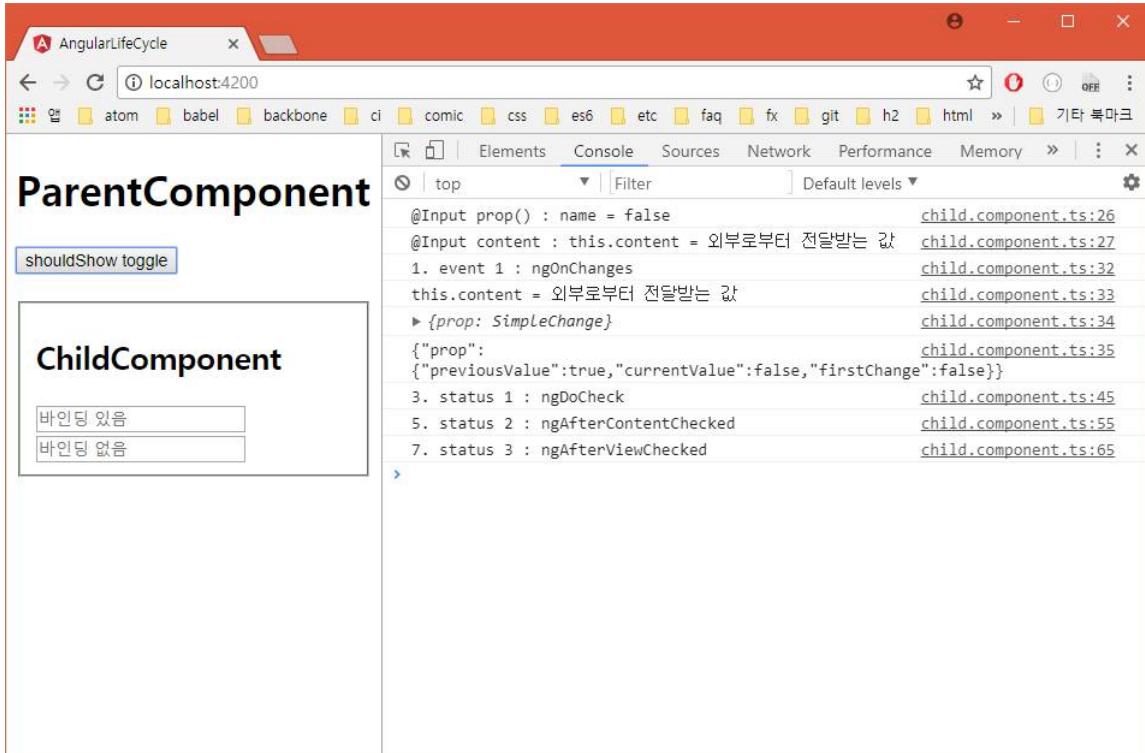
5. status 2 : ngAfterContentChecked

7. status 3 : ngAfterViewChecked

생성자에서 컴포넌트와 관련한 작업을 할 수 있는 방법이 존재하지 않는다. 생성자는 새 객체의 멤버 프로퍼티를 설정하는 용도로만 사용한다.

라이프 사이클 흐름 메소드들 보다 앞서서 @Input 데코레이터로 설정된 부분이 먼저 처리된다. 따라서, ngOnChanges 메소드에서 this.content 값을 이용할 수 있다. ngOnChanges 메소드는 파라미터로 SimpleChange 자료형을 갖는 객체를 받는다.

다음은 "shouldShow toggle" 버튼을 클릭한 결과이다. 편의상 이전 로그는 사전에 지웠다.



이미 ChildComponent는 화면에 존재하므로 생성자는 기동하지 않는다. 부모 컴포넌트로부터 전달되는 값이 변경되었으므로 자식 컴포넌트의 @Input 데코레이터로 설정된 부분이 처리된다. **ngOnChanges 메소드로 전달되는 파라미터의 값은 변경된 부분만이 포함된다.** 따라서, 파라미터로 받은 객체의 구성은 다음과 같이 변화되었다.

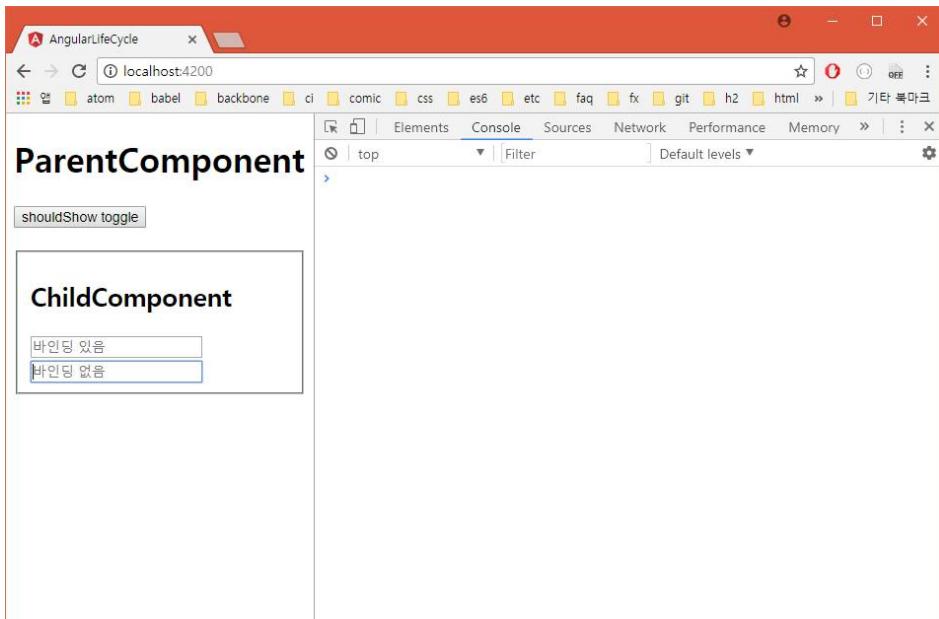
```
{content: SimpleChange, prop: SimpleChange}  
{"content":{"currentValue":"외부로부터 전달받는 값", "firstChange":true}, "prop":{"currentValue":true, "firstChange":true}}
```



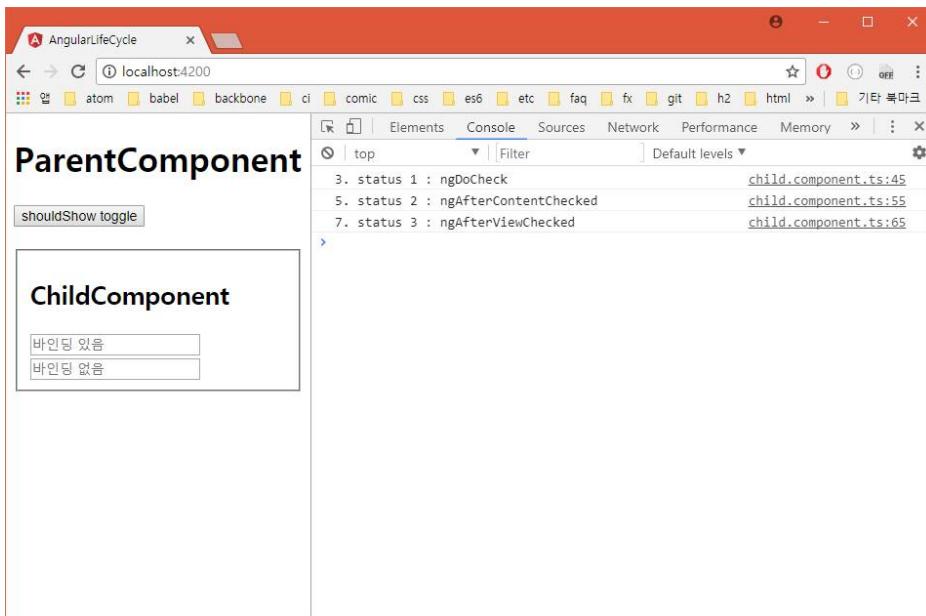
```
{prop: SimpleChange}  
{"prop":{"previousValue":true, "currentValue":false, "firstChange":false}}
```

브라우저의 콘솔창에 로그를 지운다.

"바인딩 없음" 이 표시된 인풋박스에 포커스를 주었다가 없애보자. 아무런 로그가 찍히지 않는다.



이번에는 "바인딩 있음" 이 표시된 인풋박스에 포커스를 주었다가 없애보자.



변경감지가 작동했고 이 때 status 헥 메소드들을 사용할 수 있음을 알 수 있다.

### ngOnChanges

입력 또는 출력 바인딩 값이 변경되면 호출됩니다. 바인딩이 **실제로 상태를 변경 한 경우에만** 기동됩니다.

### ngOnInit

ngOnChanges 이후에 기동합니다. 컴포넌트가 생성될 때 한 번만 수행됩니다. 컴포넌트 생성 직후에 복잡한 초기화 작업을 수행 할 장소입니다.

### ngDoCheck

이는 변경 감지를 트리거 할 수있는 항목(예 : 클릭 핸들러, http 요청, 경로 변경 등)이 발생할 때마다 시작됩니다. 이 라이프 사이클 흐름은 주로 디버그 목적으로 사용됩니다. 여기에 로직을 배치하는 것은 비용이 매우 크다는 것을 기억하세요.

### ngAfterContentInit

외부 콘텐츠가 구성 요소에 투영 된 후에 호출됩니다. @ContentChildren 및 @ContentChild와 같은 콘텐츠 쿼리는 혹 콜백이 호출되기 전에 설정된다는 점을 기억하십시오.

### ngAfterContentChecked

구성 요소 내용을 점검 할 때마다 호출됩니다.

### ngAfterViewInit

컴포넌트의 뷰가 초기화 된 후 기동됩니다. 적용 작업을 수행하기 전에 뷰가 구성되어 있어야만 하는 이벤트관련 타사 라이브러리를 초기화하는 데 이상적입니다.

### ngAfterViewChecked

컴포넌트의 뷰의 모든 변경감지 대상을 체크한 후에 호출됩니다.

### ngOnDestroy

구성 요소가 파괴되기 바로 전에 호출됩니다. 더 이상 사용하지 않는 모든 jQuery 및 서드파티 라이브러리 관련 리소스를 해제하기에 적당합니다.

### ngAfterContentInit vs ngAfterViewInit

@ContentChildren과 같은 쿼리기능을 사용하려는 경우 ngAfterContentInit 를 사용하십시오. 컨텐츠 투영은 구성 요소 외부에서 HTML 내용을 가져 와서 해당 내용을 지정된 지점에서 컴포넌트의 템플릿에 삽입하는 방법입니다.

대신 뷰를 jQuery 플러그인이나 간단한 외부 DOM 조작과 같이 랜더링해야하는 초기화 작업에는 ngAfterViewInit 를 사용하면 대부분의 다른 후크에서 setTimeout을 사용하지 않아도됩니다.

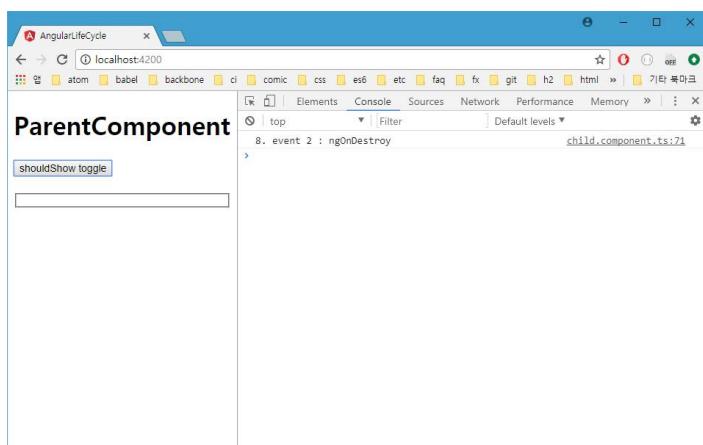
### ngAfterContentChecked vs ngAfterViewChecked

두 후크는 신중하게 사용해야하며 변경 감지가 발생할 때마다 이러한 후크가 호출되기 때문에 간단한 구현에만 사용해야 합니다.

컴포넌트가 화면에서 삭제될 때 해당 라이프사이클 메소드가 기동하는지 확인해 보자.

#### parent.component.html

```
<h1>{{title}}</h1>
<button (click)="toggle()">shouldShow toggle</button>
<br /><br />
<fieldset>
  <!-- <app-child> </app-child> -->
  <app-child [prop]="shouldShow" [content]=""외부로부터 전달받는 값"" *ngIf="shouldShow"></app-child>
</fieldset>
```



버튼을 클릭하면 자식 컴포넌트가 화면에서 사라지고 그 전에 ngOnDestroy 메소드가 기동함을 확인할 수 있다.

## Step 2 – Component Communication

### @Input and @Output

```
ng g component book
```

#### book.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {
  book;
  books = [
    { id: '001', name: '타샤의 정원 (리커버 특별판)', price: 15800, date: '20170831', img: 'img001.jpg' },
    { id: '002', name: '언어의 온도', price: 13800, date: '20160801', img: 'img002.jpg' },
    { id: '003', name: '보노보노처럼 살다니 다행이야 (특별판)', price: 16000, date: '20170401', img: 'img003.jpg' },
    { id: '004', name: '청춘의 독서 (리커버 에디션)', price: 14800, date: '20170701', img: 'img004.jpg' },
    { id: '005', name: '나는 나로 살기로 했다', price: 13800, date: '20161101', img: 'img005.jpg' }];
  constructor() {}

  ngOnInit() {}

  onSelectBook(name) {
    this.book = this.books.find(book => book.name === name);
    var year = this.book.date.substring(0, 4);
    var month = this.book.date.substring(4, 6);
    var day = this.book.date.substring(6, 8);
    this.book.publishDate = year + '-' + month + '-' + day;
  }
}
```

## book.component.html

```
<h3>도서리스트</h3>
<div *ngFor="let book of books">
  <app-book-image [path]="book.img" [title]="book.name" [width]="80" [height]="120"
    (selectBook)="onSelectBook(book.name)"></app-book-image>
</div>
<br>
<span *ngIf="book">
  <hr>
  <h3>선택된 도서</h3>
  <div>{{book.name}}</div>
  <div>{{book.price | number}}</div>
  <div>{{book.publishDate}}</div>
</span>
```

```
ng g component book/book-image
```

## book-image.component.ts

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'app-book-image',
  templateUrl: './book-image.component.html',
  styleUrls: ['./book-image.component.css']
})
export class BookImageComponent implements OnInit {
  @Input() path = "./assets/image/img001.jpg";
  @Input() title = "제목";
  @Input() width = "100";
  @Input() height = "150";
  @Output() selectBook = new EventEmitter<string>();

  constructor() {}

  ngOnInit() {}

  onClick(name: string) {
    this.selectBook.emit(name);
  }
}
```

```
 }  
  
 }
```

### book-image.component.html

```
<a (click)="onClick(title)">  
   {{title}}  
</a>
```

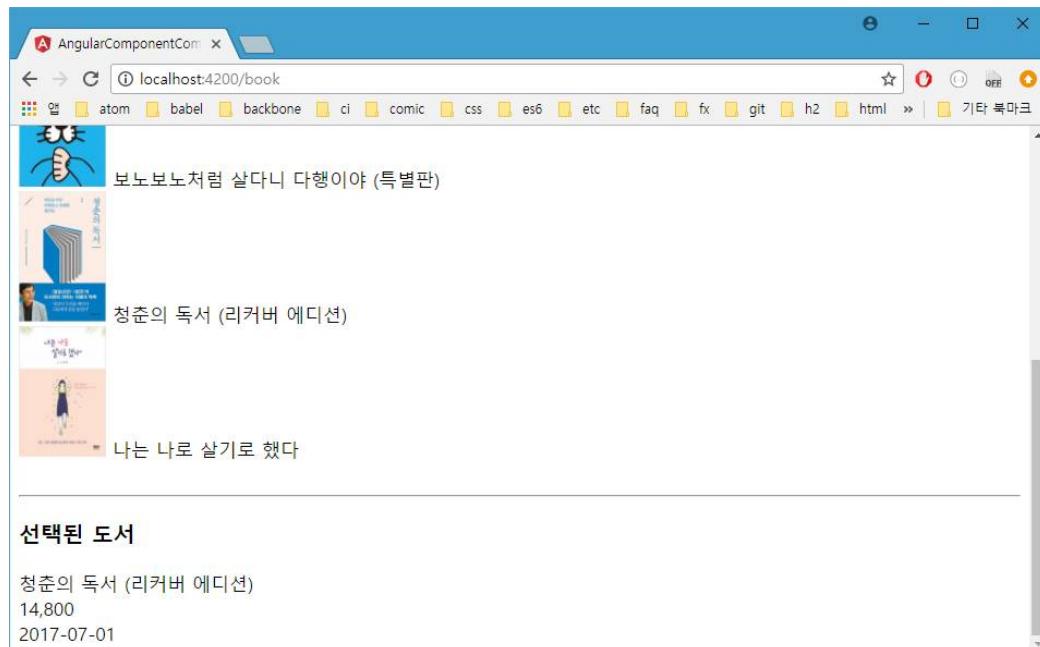
### app-routing.module.ts

```
import { NgModule } from '@angular/core';  
import { Routes, RouterModule } from '@angular/router';  
import { BookComponent } from './book/book.component';  
  
const routes: Routes = [  
  { path: '', redirectTo: '/book', pathMatch: 'full' },  
  { path: 'book', component: BookComponent}  
];  
  
@NgModule({  
  imports: [RouterModule.forRoot(routes)],  
  exports: [RouterModule]  
})  
export class AppRoutingModule { }
```

### app.component.html

```
<div class="bs-example">  
  <ul class="nav nav-pills">  
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">  
      <a routerLink="book">Book</a>  
    </li>  
    <li class="dropdown" routerLinkActive="active">  
      <a href="#" data-toggle="dropdown" class="dropdown-toggle"># <b class="caret"></b></a>  
      <ul class="dropdown-menu">  
        <li><a routerLink="#">#</a></li>  
        <li class="divider"></li>  
        <li><a routerLink="#">#</a></li>  
      </ul>  
    </li>  
  </ul>
```

```
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>
```



## @ViewChild

```
ng g component view-child
```

### view-child.component.ts

```
import { Component, OnInit } from '@angular/core';
import { AfterViewInit, Directive, Input, ViewChild } from '@angular/core';

@Directive({
  selector: 'item'
})
export class Item {
  @Input() status: boolean;
}

@Component({
  selector: 'item-component',
  template: `<button>알림창</button>`
})
export class ItemComponent {
  display(str: string) {
    alert('ItemComponent # display() : str = ' + str);
  }
}

@Component({
  selector: 'app-view-child',
  // templateUrl: './view-child.component.html',
  template: `
    <item status="true" *ngIf="isShow==true"></item>
    <item status="false" *ngIf="isShow==false"></item>
    <button (click)="toggle()">toggle</button> <br>
    isShow : {{isShow}}<br>
    status : {{status}}<br>
    <fieldset>
      <item-component (click)="display()"></item-component>
    </fieldset>`,
  styleUrls: ['./view-child.component.css']
})
export class ViewChildComponent implements OnInit {
```

```

status: boolean;

// 처음 실행할 때 isShow: boolean = true; 이므로
// <item status="true" *ngIf="isShow==true"></item> 가 선택된다.
// 따라서 item �렉티브의 status 값은 true로 초기에 설정된다.
// 토글 함수가 호출되면 isShow 값이 반전된다.
isShow: boolean = true;

// @ViewChild(접근할 컴포넌트의 클래스명)
// @ViewChild 데코레이터를 사용해서 �렉티브인 Item 자원에 접근한다.
@ViewChild(Item)
set item(d: Item) {
  console.log(`@ViewChild(Item) set item(item: Item) called.`);
  // 디렉티브의 객체는 화면이 초기화되고 나서야 접근할 수 있기 때문에
  // setTimeout 함수를 사용하여 코드수행을 늦춘다.
  setTimeout(() => { this.status = d.status; }, 0);
}

@ViewChild(ItemComponent) itemComponent: ItemComponent;

constructor() {}

ngOnInit() {
  console.log('ViewChildComponent # ngOnInit');
}

toggle() {
  alert('ViewChildComponent # toggle');
  this.isShow = !this.isShow;
}

display() {
  alert('ViewChildComponent # display');
  this.itemComponent.display('sended message from ViewChildComponent');
}
}

```

## app.module.ts

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

```

```

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { BookImageComponent } from './book/book-image/book-image.component';
import { ViewChildComponent, Item, ItemComponent } from './view-child/view-child.component';

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  declarations: [
    AppComponent,
    BookComponent, BookImageComponent,
    ViewChildComponent, Item, ItemComponent
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

### app-routing.module.ts

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { BookComponent } from './book/book.component';
import { ViewChildComponent } from './view-child/view-child.component';

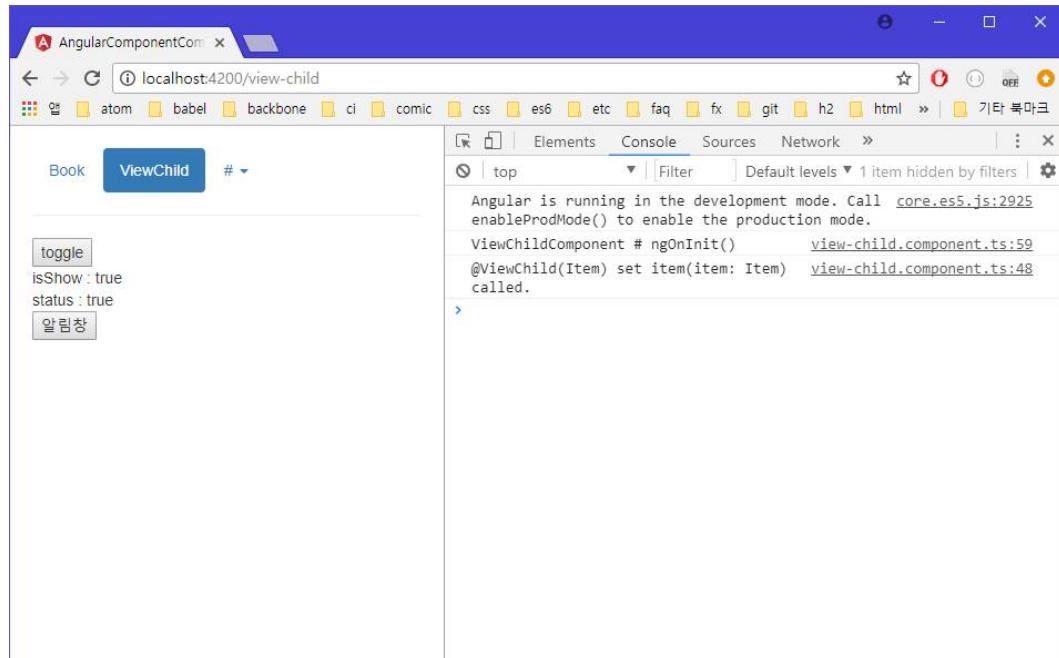
const routes: Routes = [
  { path: "", redirectTo: '/book', pathMatch: 'full' },
  { path: 'book', component: BookComponent},
  { path: 'view-child', component: ViewChildComponent},
];

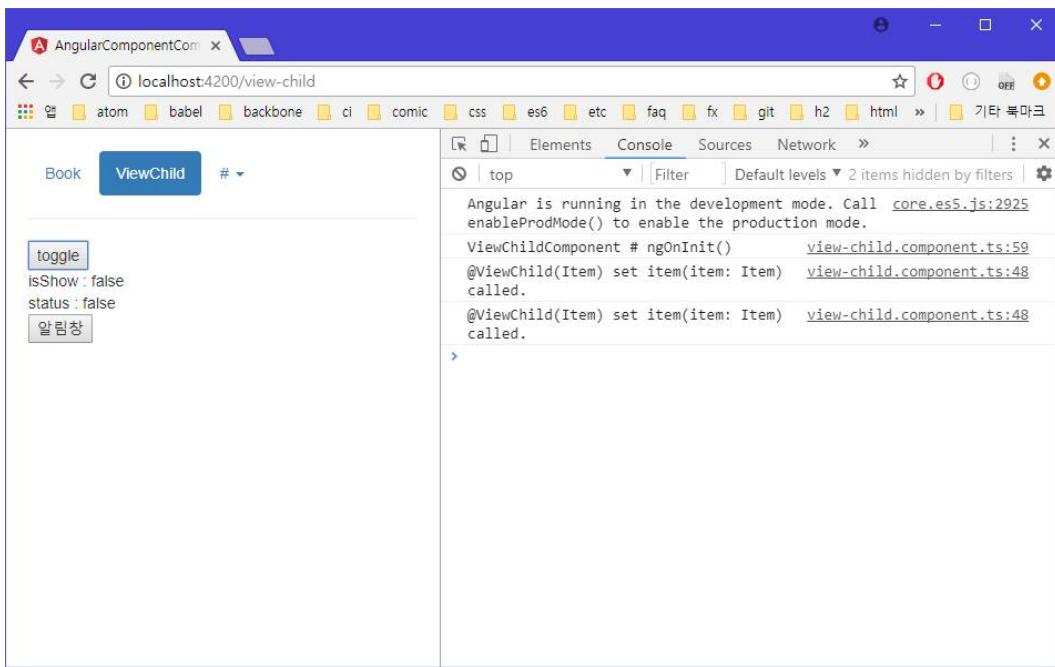
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

## app.component.html

```
<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="book">Book</a>
    </li>
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="view-child">ViewChild</a>
    </li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle"># <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="#">#</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet></router-outlet>
  </div>
</div>
```





## Observable & Subject

대화해야 하는 두개의 컴포넌트가 교대로 화면을 차지한다고 가정해 보자. 이 때는 서비스를 사용해 두 개의 컴포넌트가 대화할 수 있다. 컴포넌트 전환 시 라이프사이클 메소드 중 `ngOnInit()`가 기동하므로 초기화 작업으로 서비스로부터 데이터를 가져오면 된다.

그런데 만약 두 개의 컴포넌트가 동시에 화면에 띠 있는 상태라면 초기화 메소드를 이용해서는 다이나믹하게 데이터를 주고 받을 수 없게 된다. 다른 대안이 필요하다. 지금부터 rxjs의 비동기 기술을 통해 처리하는 방법을 살펴보자.

A 컴포넌트 : `publish → Observable ← subscribe : B 컴포넌트`

Observable 객체는 `EventEmitter` 객체와 유사하게 작동한다. A 컴포넌트에서 이벤트가 발생하여 데이터가 변경되면 이를 Observable에게 알린다. 그런 다음 Observable은 구독신청을 한 B 컴포넌트의 콜백함수를 호출함으로써 이를 연동하는 방식이다.

```
ng g component counter
ng g component counter/counter-display
ng g component counter/counter-control
```

### counter-display.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-counter-display',
  templateUrl: './counter-display.component.html',
  styleUrls: ['./counter-display.component.css']
})
export class CounterDisplayComponent implements OnInit {
  count = 0;

  constructor() { }

  ngOnInit() {
  }
}
```

### **counter-display.component.html**

```
<div class="">
  <h2>count: {{count}}</h2>
</div>
```

### **counter-control.component.ts**

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-counter-control',
  templateUrl: './counter-control.component.html',
  styleUrls: ['./counter-control.component.css']
})
export class CounterControlComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

  increment() {
    alert('CounterDisplayComponent 의 count 값을 +1 한다.');
  }

  decrement() {
    alert('CounterDisplayComponent 의 count 값을 -1 한다.');
  }
}
```

### **counter-control.component.html**

```
<div class="">
  <button type="button" (click)="increment()">increment</button>
  <button type="button" (click)="decrement()">decrement</button>
</div>
```

### **counter.component.ts**

```
import { Component, OnInit } from '@angular/core';
```

```

@Component({
  selector: 'app-counter',
  templateUrl: './counter.component.html',
  styleUrls: ['./counter.component.css']
})
export class CounterComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}

```

### **counter.component.html**

```

<div class="panel panel-default">
  <div class="panel-body">
    <app-counter-display></app-counter-display>
  </div>
</div>
<div class="panel panel-default">
  <div class="panel-body">
    <app-counter-control></app-counter-control>
  </div>
</div>

```

### **app-routing.module.ts**

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { BookComponent } from './book/book.component';
import { ViewChildComponent } from './view-child/view-child.component';
import { CounterComponent } from './counter/counter.component';

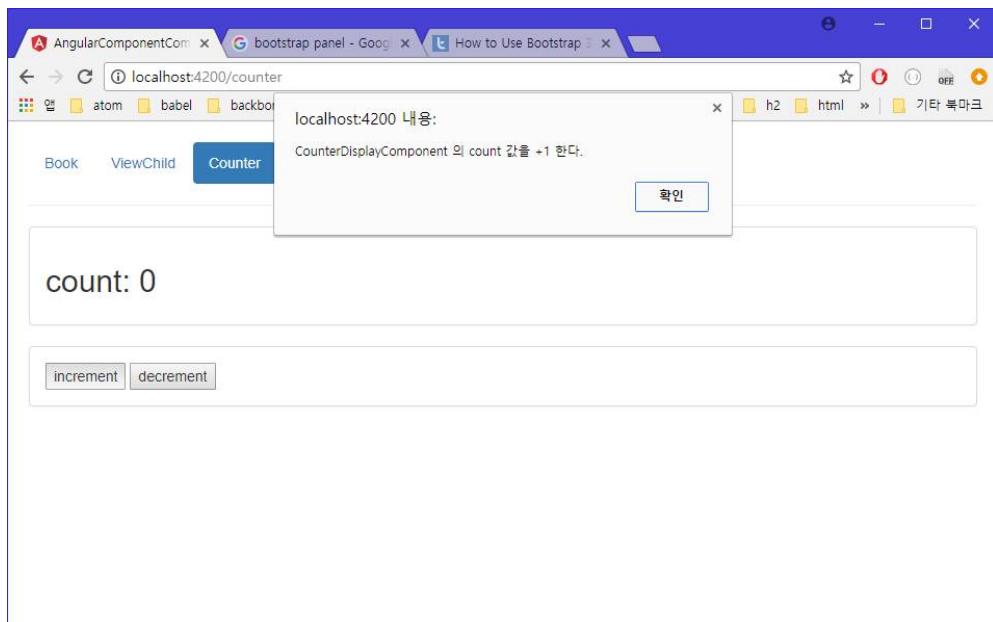
const routes: Routes = [
  { path: '', redirectTo: '/book', pathMatch: 'full' },
  { path: 'book', component: BookComponent},
  { path: 'counter', component: CounterComponent},
  { path: 'view-child', component: ViewChildComponent},
];

```

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

### app.component.html

```
<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="book">Book</a>
    </li>
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="view-child">ViewChild</a>
    </li>
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="counter">Counter</a>
    </li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle"># <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="#">#</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet></router-outlet>
  </div>
</div>
```



```
ng g service counter/counter-bridge
```

### counter-bridge.service.ts

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { Subject } from 'rxjs/Subject';

@Injectable()
export class CounterBridgeService {
  private subject = new Subject<any>();

  constructor() {}

  increment() {
    this.subject.next({ type: 'increment' });
  }

  decrement() {
    this.subject.next({ type: 'decrement' });
  }

  getObservable(): Observable<any> {
    return this.subject.asObservable();
  }
}
```

```
}
```

### app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { BookImageComponent } from './book/book-image/book-image.component';
import { ViewChildComponent, Item, ItemComponent } from './view-child/view-child.component';
import { CounterComponent } from './counter/counter.component';
import { CounterDisplayComponent } from './counter/counter-display/counter-display.component';
import { CounterControlComponent } from './counter/counter-control/counter-control.component';

import { CounterBridgeService } from './counter/counter-bridge.service';

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  declarations: [
    AppComponent,
    BookComponent, BookImageComponent,
    ViewChildComponent, Item, ItemComponent,
    CounterComponent, CounterDisplayComponent, CounterControlComponent
  ],
  providers: [CounterBridgeService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

### counter-display.component.ts

```
import { Component, OnInit } from '@angular/core';
import { CounterBridgeService } from './counter-bridge.service';

@Component({
  selector: 'app-counter-display',
  templateUrl: './counter-display.component.html',
})
```

```

        styleUrls: ['./counter-display.component.css']
    })
}

export class CounterDisplayComponent implements OnInit {
    count = 0;

    constructor(private counterBridgeService: CounterBridgeService) { }

    ngOnInit() {
        this.counterBridgeService.getObservable().subscribe(
            message => {
                if (message.type === 'incremnet') {
                    this.count++;
                } else {
                    this.count--;
                }
            }
        );
    }
}

```

### counter-control.component.ts

```

import { Component, OnInit } from '@angular/core';
import { CounterBridgeService } from './counter-bridge.service';

@Component({
    selector: 'app-counter-control',
    templateUrl: './counter-control.component.html',
    styleUrls: ['./counter-control.component.css']
})
export class CounterControlComponent implements OnInit {

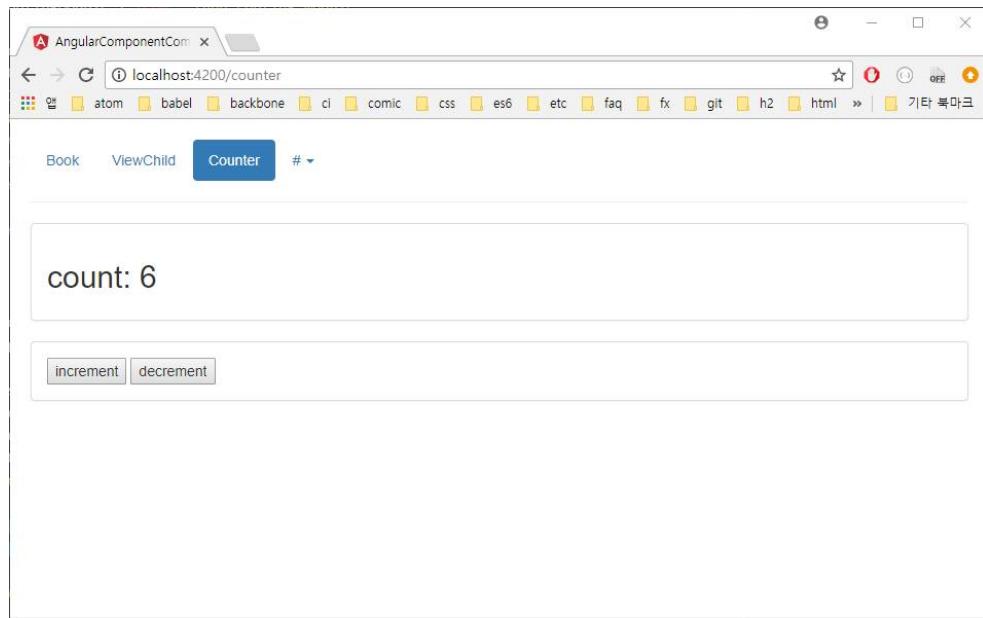
    constructor(private counterBridgeService: CounterBridgeService) { }

    ngOnInit() {
    }

    increment() {
        // alert('CounterDisplayComponent 의 count 값을 +1 한다.');
        this.counterBridgeService.incremnet();
    }
}

```

```
decrement() {
    // alert('CounterDisplayComponent 의 count 값을 -1 한다.');
    this.counterBridgeService.decrement();
}
}
```



## @ContentChild

```
ng g component content-child
```

### content-child.component.ts

```
import { Component, OnInit, ContentChild, Directive, Input } from '@angular/core';

@Directive({ selector: 'pane' })
export class Pane {
    @Input() id: string;
}

@Component({
    selector: 'tab',
    template: `<div>pane: {{pane?.id}}</div>`
})
export class Tab {
    @ContentChild(Pane) pane: Pane;
}

@Component({
    selector: 'app-content-child',
    // templateUrl: './content-child.component.html',
    template: `
        <tab>
            <pane id="1" *ngIf="shouldShow"></pane>
            <pane id="2" *ngIf="!shouldShow"></pane>
        </tab>
        <button (click)="toggle()">Toggle</button>
    `,
    styleUrls: ['./content-child.component.css']
})
export class ContentChildComponent implements OnInit {
    shouldShow = true;

    constructor() { }

    ngOnInit() {
    }
}
```

```

toggle() {
  this.shouldShow = !this.shouldShow;
}
}

```

Tab 컴포넌트가 자신의 템플릿에서 직접 Pane 디렉티브를 사용했다면 @ViewChild로 접근한다.

앞 예제와 같이, ContentChildComponent 컴포넌트의 템플릿 설정에 의해서 Tab 컴포넌트가 Pane 디렉티브를 자식으로 갖게 되는 경우 @ContentChild로 접근한다.

### app.module.ts

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { BookImageComponent } from './book/book-image/book-image.component';
import { ViewChildComponent, Item, ItemComponent } from './view-child/view-child.component';
import { CounterComponent } from './counter/counter.component';
import { CounterDisplayComponent } from './counter/counter-display/counter-display.component';
import { CounterControlComponent } from './counter/counter-control/counter-control.component';

import { CounterBridgeService } from './counter/counter-bridge.service';
import { ContentChildComponent, Pane, Tab } from './content-child/content-child.component';

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  declarations: [
    AppComponent,
    BookComponent, BookImageComponent,
    ViewChildComponent, Item, ItemComponent,
    CounterComponent, CounterDisplayComponent, CounterControlComponent,
    ContentChildComponent, Pane, Tab
  ],
  providers: [CounterBridgeService],
  bootstrap: [AppComponent]
})

```

```
export class AppModule { }
```

### app.component.html

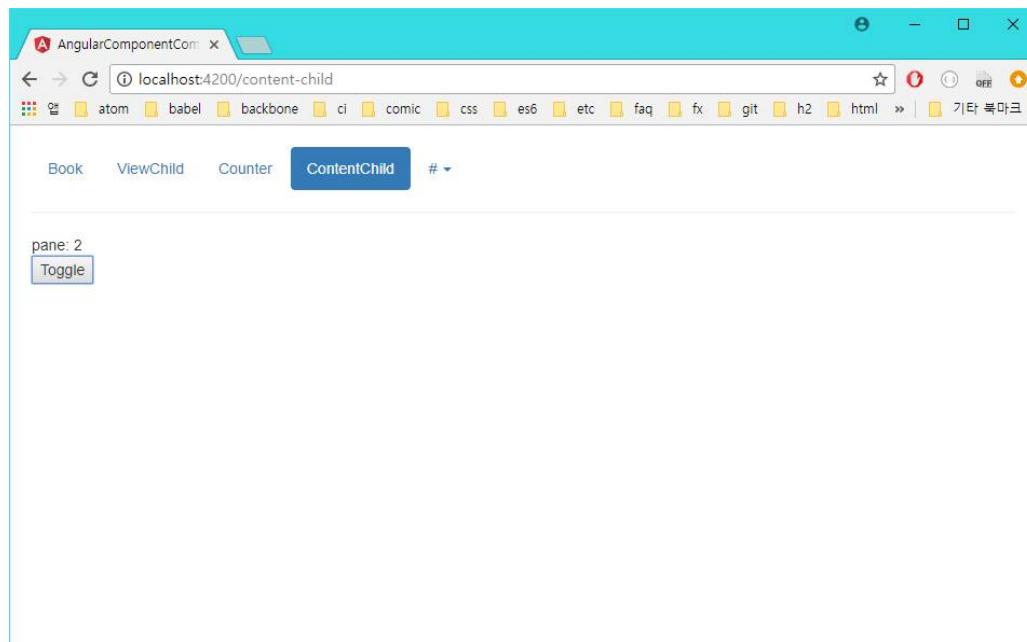
```
<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="book">Book</a>
    </li>
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="view-child">ViewChild</a>
    </li>
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="counter">Counter</a>
    </li>
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="content-child">ContentChild</a>
    </li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle"># <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="#">#</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet></router-outlet>
  </div>
</div>
```

### app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { BookComponent } from './book/book.component';
import { ViewChildComponent } from './view-child/view-child.component';
import { CounterComponent } from './counter/counter.component';
import { ContentChildComponent } from './content-child/content-child.component';
```

```
const routes: Routes = [
  { path: '', redirectTo: '/book', pathMatch: 'full' },
  { path: 'book', component: BookComponent},
  { path: 'counter', component: CounterComponent},
  { path: 'view-child', component: ViewChildComponent},
  { path: 'content-child', component: ContentChildComponent},
];
}

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```



## @ViewChildren

```
ng g component view-children
```

### view-children.component.ts

```
import { Component, OnInit, AfterViewInit, Directive, Input, QueryList, ViewChildren } from '@angular/core';
import { Pane } from './content-child/content-child.component';

@Component({
  selector: 'app-view-children',
  // templateUrl: './view-children.component.html',
  template: `
    <pane id="1"></pane>
    <pane id="2"></pane>
    <pane id="3" *ngIf="shouldShow"></pane>
    <button (click)="show()">Show 3</button>
    <div>panes: {{serializedPanes}}</div>
  `,
  styleUrls: ['./view-children.component.css']
})
export class ViewChildrenComponent implements OnInit, AfterViewInit {
  @ViewChildren(Pane) panes: QueryList<Pane>;
  serializedPanes: string = "";
  shouldShow = false;

  constructor() {}

  ngOnInit() {}

  ngAfterViewInit() {
    this.calculateSerializedPanes();
    this.panes.changes.subscribe((r) => { this.calculateSerializedPanes(); });
  }

  show() {
    this.shouldShow = true;
  }

  calculateSerializedPanes() {
```

```

    setTimeout(() => {
      this.serializedPanes = this.panes.map(p => p.id).join(', ');
    }, 0);
}

```

### app-routing.module.ts

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { BookComponent } from './book/book.component';
import { ViewChildComponent } from './view-child/view-child.component';
import { CounterComponent } from './counter/counter.component';
import { ContentChildComponent } from './content-child/content-child.component';
import { ViewChildrenComponent } from './view-children/view-children.component';

const routes: Routes = [
  { path: '', redirectTo: '/book', pathMatch: 'full' },
  { path: 'book', component: BookComponent},
  { path: 'counter', component: CounterComponent},
  { path: 'view-child', component: ViewChildComponent},
  { path: 'content-child', component: ContentChildComponent},
  { path: 'view-children', component: ViewChildrenComponent},
];
}

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

### app.component.html

```

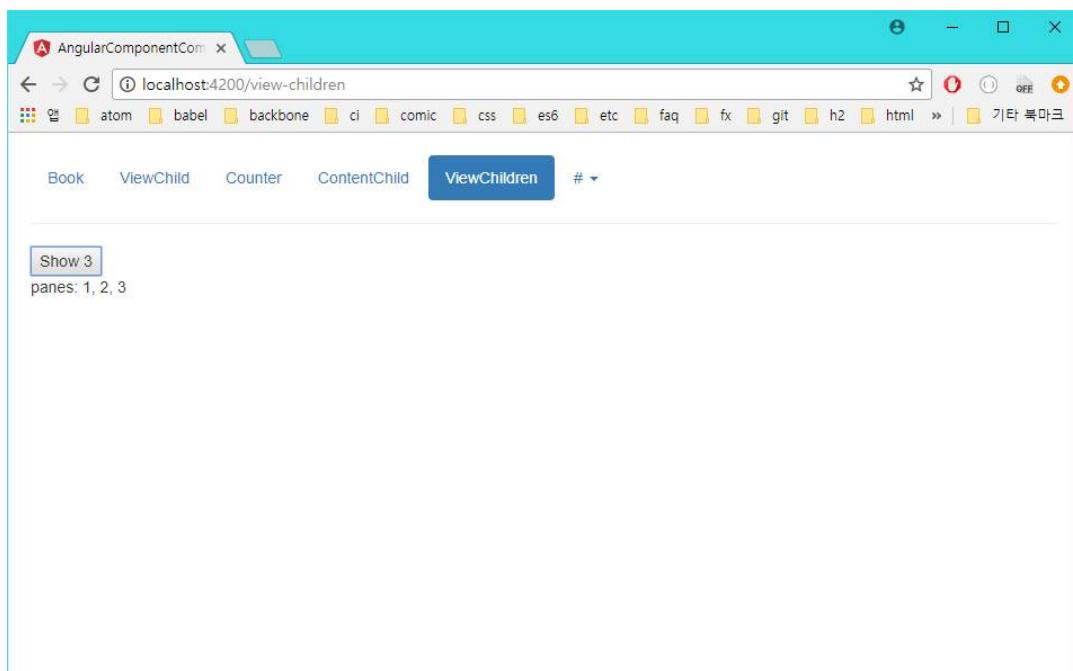
<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="book">Book</a>
    </li>
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="view-child">ViewChild</a>
    </li>
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="counter">Counter</a>
    </li>
  </ul>
</div>

```

```

</li>
<li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
    <a routerLink="content-child">ContentChild</a>
</li>
<li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
    <a routerLink="view-children">ViewChildren</a>
</li>
<li class="dropdown" routerLinkActive="active">
    <a href="#" data-toggle="dropdown" class="dropdown-toggle"># <b class="caret"></b></a>
    <ul class="dropdown-menu">
        <li><a routerLink="#">#</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
    </ul>
</li>
</ul>
<hr/>
<div class="margin">
    <router-outlet></router-outlet>
</div>
</div>

```



## json-stringify-safe

"circular structure to JSON" 에러가 발생하는 경우 다음 기술을 사용해 보자.

```
npm install json-stringify-safe --save
```

### view-children.component.ts

```
import { Component, OnInit, AfterViewInit, Directive, Input, QueryList, ViewChildren } from '@angular/core';
import { Pane } from './content-child/content-child.component';

import stringify from 'json-stringify-safe';

@Component({
  selector: 'app-view-children',
  // templateUrl: './view-children.component.html',
  template: `
    <pane id="1"></pane>
    <pane id="2"></pane>
    <pane id="3" *ngIf="shouldShow"></pane>
    <button (click)="show()">Show 3</button>
    <div>panes: {{serializedPanes}}</div>
  `,
  styleUrls: ['./view-children.component.css']
})
export class ViewChildrenComponent implements OnInit, AfterViewInit {
  @ViewChildren(Pane) panes: QueryList<Pane>;
  serializedPanes: string = "";
  shouldShow = false;

  constructor() { }

  ngOnInit() {
  }

  ngAfterViewInit() {
    this.calculateSerializedPanes();
    this.panes.changes.subscribe((r) => {
      console.log(JSON.stringify(stringify(r)));
      this.calculateSerializedPanes();
    });
  }

  show() {
```

```

        this.shouldShow = true;
    }

    calculateSerializedPanes() {
        setTimeout(() => {
            this.serializedPanes = this.panes.map(p => p.id).join(', ');
        }, 0);
    }
}

```

```

{
    "_dirty": false,
    "_results": [{"id": "1"}, {"id": "2"}, {"id": "3"}],
    "_emitter": {"_isScalar": false,
        "observers": [{"closed": false, "_parent": null, "_parents": null, "_subscriptions": [{"closed": false, "_parent": null, "_parents": null, "subject": [
            Circular~._emitter.observers.0
        ], "_parents": null, "_subscriptions": null, "syncErrorValue": null, "syncErrorThrown": false, "syncErrorThrowable": false, "isStopped": false, "destination": {"closed": false, "_parent": null, "_parents": null, "syncErrorValue": null, "syncErrorThrown": false, "syncErrorThrowable": false, "isStopped": false, "destination": {"closed": true, "_parentSubscriber": [
            Circular~._emitter.observers.0
        ]}, "context": [
            Circular~._emitter.observers.0.destination
        ]}}, "closed": false, "isStopped": false, "hasError": false, "thrownError": null, "__isAsync": false}]}
}

```

위 내용을 온라인 JSON 뷰어로 보면 다음과 같다.

The screenshot shows the Online JSON Viewer interface with the URL [jsonviewer.stack.hu](http://jsonviewer.stack.hu). The JSON structure is displayed in two panes:

- Left pane (Viewer):** Shows the hierarchical structure of the JSON object. It includes nodes for `_dirty`, `_results` (containing three items with IDs 1, 2, 3), `_emitter` (containing `_isScalar`, `observers` (containing `closed`, `isStopped`, `hasError`, `thrownError`, and `__isAsync`)), and a table view of the `_results` array.
- Right pane (Table):** A table showing the values for the `_results` array. The columns are `Name` and `Value`. The data is as follows:

Name	Value
<code>_dirty</code>	false
<code>_emitter</code>	...
<code>_results</code>	...

## @ContentChildren

```
ng g component content-children
```

### content-children.component.ts

```
import { Component, OnInit, ContentChildren, Directive, Input, QueryList } from '@angular/core';
import { Pane } from './content-child/content-child.component';

@Component({
  selector: 'index',
  template: `
    <div class="top-level">Top level panes: {{serializedPanes}}</div>
    <div class="nested">Arbitrary nested panes: {{serializedNestedPanes}}</div>
  `
})
export class Index {
  @ContentChildren(Pane) topLevelPanes: QueryList<Pane>;
  // Index 컴포넌트가 바로 밑으로 갖고 있는 자식 엘리먼트뿐만 아니라
  // Index 컴포넌트가 중첩된 환경에서 자식의 자식 엘리먼트도 참조한다.
  @ContentChildren(Pane, { descendants: true }) arbitraryNestedPanes: QueryList<Pane>;

  get serializedPanes(): string {
    return this.topLevelPanes ? this.topLevelPanes.map(p => p.id).join(' ') : '';
  }
  get serializedNestedPanes(): string {
    return this.arbitraryNestedPanes ? this.arbitraryNestedPanes.map(p => p.id).join(' ') : '';
  }
}

@Component({
  selector: 'app-content-children',
  // templateUrl: './content-children.component.html',
  template: `
    <index>
      <pane id="1"></pane>
      <pane id="2"></pane>
      <pane id="3" *ngIf="shouldShow">
        <index>
          <pane id="3.1"></pane>
          <pane id="3.2"></pane>
        </index>
      </pane>
    </index>
  `
})
```

```

        </index>
    </pane>
</index>
<button (click)="toggle()">Toggle 3</button>
';
styleUrls: ['./content-children.component.css']
})
export class ContentChildrenComponent implements OnInit {
shouldShow = false;

constructor() { }

ngOnInit() {
}

toggle() {
    this.shouldShow = !this.shouldShow;
}
}

```

### app.component.html

```

<div class="bs-example">
<ul class="nav nav-pills">
<li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
    <a routerLink="book">Book</a>
</li>
<li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
    <a routerLink="view-child">ViewChild</a>
</li>
<li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
    <a routerLink="counter">Counter</a>
</li>
<li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
    <a routerLink="content-child">ContentChild</a>
</li>
<li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
    <a routerLink="view-children">ViewChildren</a>
</li>
<li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
    <a routerLink="content-children">ContentChildren</a>
</li>

```

```

<li class="dropdown" routerLinkActive="active">
  <a href="#" data-toggle="dropdown" class="dropdown-toggle"># <b class="caret"></b></a>
  <ul class="dropdown-menu">
    <li><a routerLink="#" #></a></li>
    <li class="divider"></li>
    <li><a routerLink="#" #></a></li>
  </ul>
</li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>

```

### app-routing.module.ts

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { BookComponent } from './book/book.component';
import { ViewChildComponent } from './view-child/view-child.component';
import { CounterComponent } from './counter/counter.component';
import { ContentChildComponent } from './content-child/content-child.component';
import { ViewChildrenComponent } from './view-children/view-children.component';
import { ContentChildrenComponent } from './content-children/content-children.component';

const routes: Routes = [
  { path: '', redirectTo: '/book', pathMatch: 'full' },
  { path: 'book', component: BookComponent},
  { path: 'counter', component: CounterComponent},
  { path: 'view-child', component: ViewChildComponent},
  { path: 'content-child', component: ContentChildComponent},
  { path: 'view-children', component: ViewChildrenComponent},
  { path: 'content-children', component: ContentChildrenComponent},
];
}

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

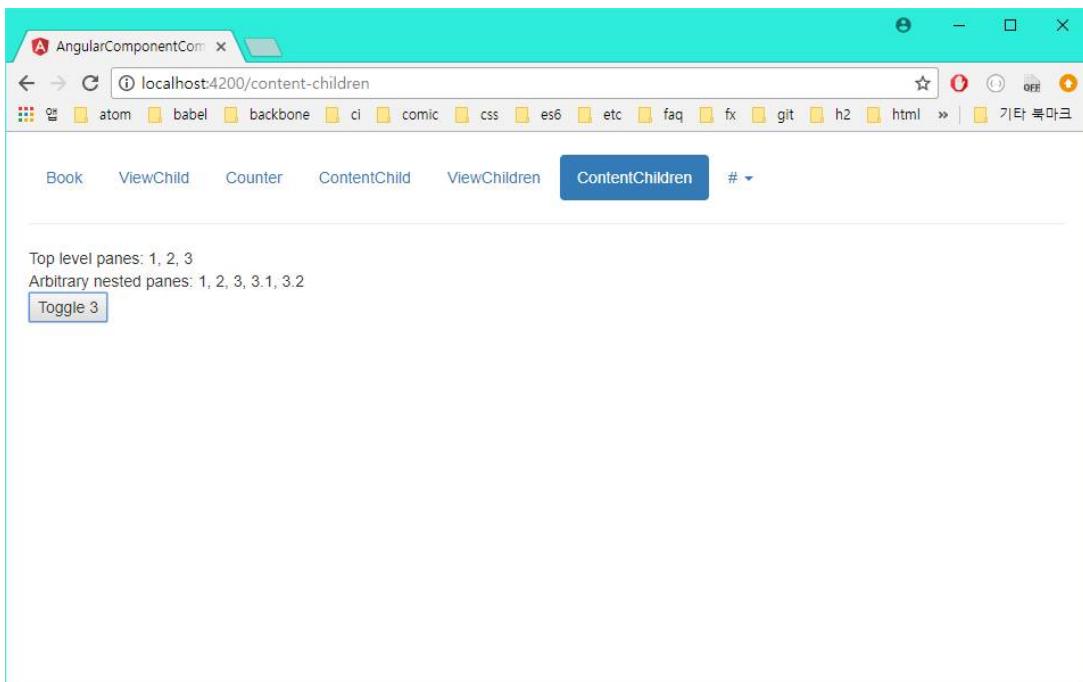
## app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { BookImageComponent } from './book/book-image/book-image.component';
import { ViewChildComponent, Item, ItemComponent } from './view-child/view-child.component';
import { CounterComponent } from './counter/counter.component';
import { CounterDisplayComponent } from './counter/counter-display/counter-display.component';
import { CounterControlComponent } from './counter/counter-control/counter-control.component';

import { CounterBridgeService } from './counter/counter-bridge.service';
import { ContentChildComponent, Pane, Tab } from './content-child/content-child.component';
import { ViewChildrenComponent } from './view-children/view-children.component';
import { ContentChildrenComponent, Index } from './content-children/content-children.component';

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  declarations: [
    AppComponent,
    BookComponent, BookImageComponent,
    ViewChildComponent, Item, ItemComponent,
    CounterComponent, CounterDisplayComponent, CounterControlComponent,
    ContentChildComponent, Pane, Tab,
    ViewChildrenComponent,
    ContentChildrenComponent, Index
  ],
  providers: [CounterBridgeService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



# Step 3 – HTTP

## Promise

```
ng new angular-http-example --routing=true
```

### index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>AngularHttpExample</title>
    <base href="/">

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
    <style type="text/css">
      .bs-example {
        margin: 20px;
      }
    </style>

  </head>
  <body>
    <app-root></app-root>
  </body>
</html>
```

```
ng g component kpop
ng g service kpop/kpop-http
cd src/app/kpop && echo '' > kpop.model.ts && cd ../../..
```

### **kpop.model.ts**

```
export class Kpop {  
    id: number;  
    name: string;  
    image: string;  
}
```

### **kpop-http.service.ts**

```
import { Injectable } from '@angular/core';  
import { Http } from "@angular/http";  
import { Kpop } from './kpop.model';  
  
import 'rxjs/add/operator/toPromise';  
  
@Injectable()  
export class KpopHttpService {  
  
    constructor(private http: Http) {}  
  
    getIdols(): Promise<Kpop[]> {  
        return this.http.get('./assets/server/kpop.json')  
            .toPromise().then(res => {  
                console.log(res);  
                return res.json().info.idols;  
            });  
    }  
}
```

### **kpop.component.ts**

```
import { Component, OnInit } from '@angular/core';  
import { KpopHttpService } from './kpop-http.service';  
import { Kpop } from './kpop.model';  
  
@Component({  
    selector: 'app-kpop',  
    templateUrl: './kpop.component.html',  
    styleUrls: ['./kpop.component.css']  
})
```

```

export class KpopComponent implements OnInit {
  idols: Kpop[];

  constructor(private kpopHttpService: KpopHttpService) { }

  ngOnInit() {
    this.kpopHttpService.getIdols().then(idols => this.idols = idols);
  }
}

```

## kpop.component.html

```

<h3>Kpop Idols</h3>
<ng-template ngFor let-idol [ngForOf]="idols">
  <div class="panel panel-default">
    <div class="panel-heading">{{idol.name}}</div>
    <div class="panel-body">
      
    </div>
  </div>
</ng-template>

```

```

cd src/assets && mkdir server && cd ../../
cd src/assets/server && echo '' > kpop.json && cd ../../..

```

## kpop.json

```

{
  "success":true,
  "info":{
    "idols":[
      {"id": 1, "name":"A-pink", "image":"img001.jpg" },
      {"id": 2, "name":"Girls Generation", "image":"img002.jpg" },
      {"id": 3, "name":"Black Pink", "image":"img003.jpg" }
    ],
  },
  "msg":"success"
}

```

## app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { FormsModule }   from '@angular/forms';
import { HttpClientModule }   from '@angular/http';

import { AppComponent } from './app.component';
import { KpopComponent } from './kpop/kpop.component';

import { KpopHttpService } from './kpop/kpop-http.service';

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule
  ],
  declarations: [
    AppComponent,
    KpopComponent
  ],
  providers: [KpopHttpService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

## app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { KpopComponent } from './kpop/kpop.component';

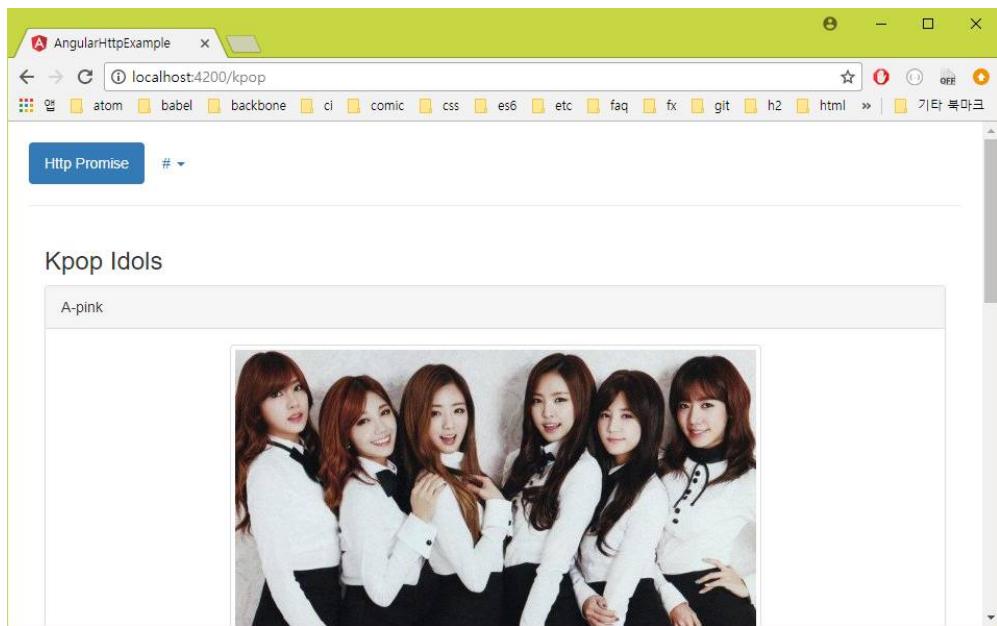
const routes: Routes = [
  { path: '', redirectTo: '/kpop', pathMatch: 'full' },
  { path: 'kpop', component: KpopComponent},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

### app.component.html

```
<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
      <a routerLink="kpop">Http Promise</a>
    </li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle"># <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="#">#</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet></router-outlet>
  </div>
</div>
```



# Observable

## 1단계 : 클라이언트 사이드 - UI

```
ng g component emp
```

### emp.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-emp',
  templateUrl: './emp.component.html',
  styleUrls: ['./emp.component.css']
})
export class EmpComponent implements OnInit {
  employees: any = [
    { id: 1, firstName: 'AA', lastName: 'aa' },
    { id: 2, firstName: 'BB', lastName: 'bb' },
    { id: 3, firstName: 'CC', lastName: 'cc' },
    { id: 4, firstName: 'DD', lastName: 'dd' },
  ];
  constructor() {}

  ngOnInit() {}

  getEmps() {}

  addEmp(firstName: string, lastName: string) {
    alert('addEmp() #' + firstName + ' ' + lastName);
  }

  removeEmp(person: any) {
    alert(JSON.stringify(person));
    return false; // anchor 태그의 이벤트 전파를 막는다.
  }

  onSubmit(f) {
```

```

if (f.valid) {
    var emp = f.value;
    console.log(emp);
    alert('onSubmit() #' + emp.firstName + ' ' + emp.lastName);
}
}
}

```

### **emp.component.html**

```

<div class="bs-example">
<div class="row">
<div class="col-md-12">
<form class="form-inline" #f="ngForm" (ngSubmit)="onSubmit(f)">
<div class="form-group">
<label class="sr-only" for="firstName">First Name</label>
<input type="text" class="form-control" [(ngModel)]="firstName"
       id="firstName" name="firstName" placeholder="First Name" required>
</div>
<div class="form-group">
<label class="sr-only" for="lastName">Last Name</label>
<input type="text" class="form-control" [(ngModel)]="lastName"
       id="lastName" name="lastName" placeholder="Last Name" required>
</div>
<!--
<button type="button" class="btn btn-default btn-sm"
[disabled]="f.invalid" (click)="addEmp(firstName, lastName)">
<b class="glyphicon glyphicon-plus"></b> Add
</button> -->

<button type="submit" class="btn btn-default btn-sm" [disabled]="f.invalid">
<b class="glyphicon glyphicon-plus"></b> Add
</button>
</form>
</div>
</div>
<br>
<div class="row">
<div *ngFor="let emp of employees" class="col-md-4">
<div class="panel panel-default">
<div class="panel-heading">{{emp.id}}</div>
<div class="panel-body">

```

```

{{emp.firstName}} {{' '}}{{emp.lastName}}

```

```

</div>
<div class="panel-footer clearfix">
    <div class="pull-right">
        <a href="#" class="text-warning" (click)="removeEmp(emp)">
            <span class="glyphicon glyphicon-remove"></span>
        </a>
    </div>
</div>
</div>
</div>
</div>

```

### app-routing.module.ts

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { KpopComponent } from './kpop/kpop.component';
import { EmpComponent } from './emp/emp.component';

const routes: Routes = [
    { path: '', redirectTo: '/kpop', pathMatch: 'full' },
    { path: 'kpop', component: KpopComponent},
    { path: 'emp', component: EmpComponent},
];

@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})
export class AppRoutingModule { }

```

### app.component.html

```

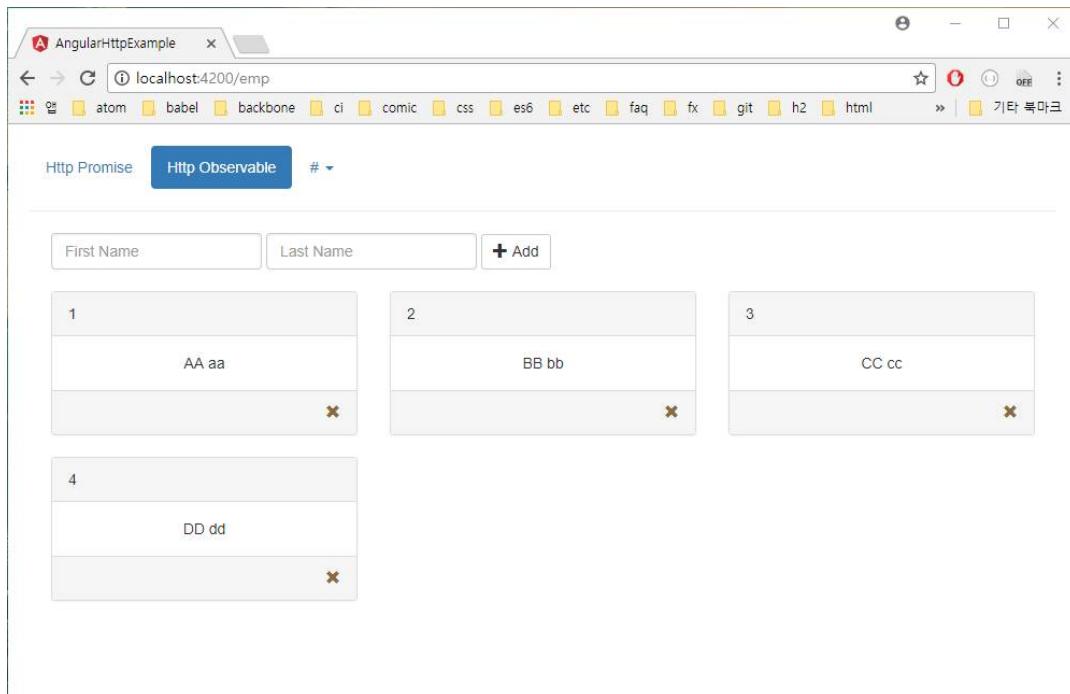
<div class="bs-example">
    <ul class="nav nav-pills">
        <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">
            <a routerLink="kpop">Http Promise</a>
        </li>
        <li routerLinkActive="active" [routerLinkActiveOptions]="{ exact: true }">

```

```

<a routerLink="emp">Http Observable</a>
</li>
<li class="dropdown" routerLinkActive="active">
  <a href="#" data-toggle="dropdown" class="dropdown-toggle"># <b class="caret"></b></a>
  <ul class="dropdown-menu">
    <li><a routerLink="#">#</a></li>
    <li class="divider"></li>
    <li><a routerLink="#">#</a></li>
  </ul>
</li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>

```



## 2단계: 서버 사이드 - Spring Framework Restful Service

STS > File > New > Spring Starter Project

프로젝트명: angular-http-server-spring

디펜던시 체크: web

클래스명 변경: AngularHttpServerSpringApplication.java > Refactor > Rename > StartApplication.java

### StartApplication.java

```
StartApplication.java package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@SpringBootApplication
public class StartApplication extends WebMvcConfigurerAdapter {

    public static void main(String[] args) {
        SpringApplication.run(StartApplication.class, args);
    }

    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addRedirectViewController("/", "/employees");
    }

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurerAdapter() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/employees/**").allowedOrigins("*")
                    .allowedMethods("POST", "GET", "PUT", "DELETE", "OPTIONS");
            }
        };
    }
}
```

```
    }  
}  
}
```

## Employee.java

```
package com.example.demo;  
  
public class Employee {  
    private int id;  
    private String firstName;  
    private String lastName;  
  
    public Employee() {}  
    public Employee(int id, String firstName, String lastName) {  
        this.id = id;  
        this.firstName = firstName;  
        this.lastName = lastName;  
    }  
  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
  
    public String getFirstName() {  
        return firstName;  
    }  
    public void setFirstName(String firstName) {  
        this.firstName = firstName;  
    }  
  
    public String getLastName() {  
        return lastName;  
    }  
    public void setLastName(String lastName) {  
        this.lastName = lastName;  
    }  
}
```

## **EmployeeRestController.java**

```
package com.example.demo;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EmployeeRestController {
    private List<Employee> employees = new ArrayList<Employee>();

    public EmployeeRestController() {
        employees.add(new Employee(1, "ADAM", "Sandler"));
        employees.add(new Employee(2, "BOB", "Ross"));
        employees.add(new Employee(3, "CHRIS", "Evans"));
    }

    @GetMapping("/employees")
    public Object get() {
        return employees;
    }

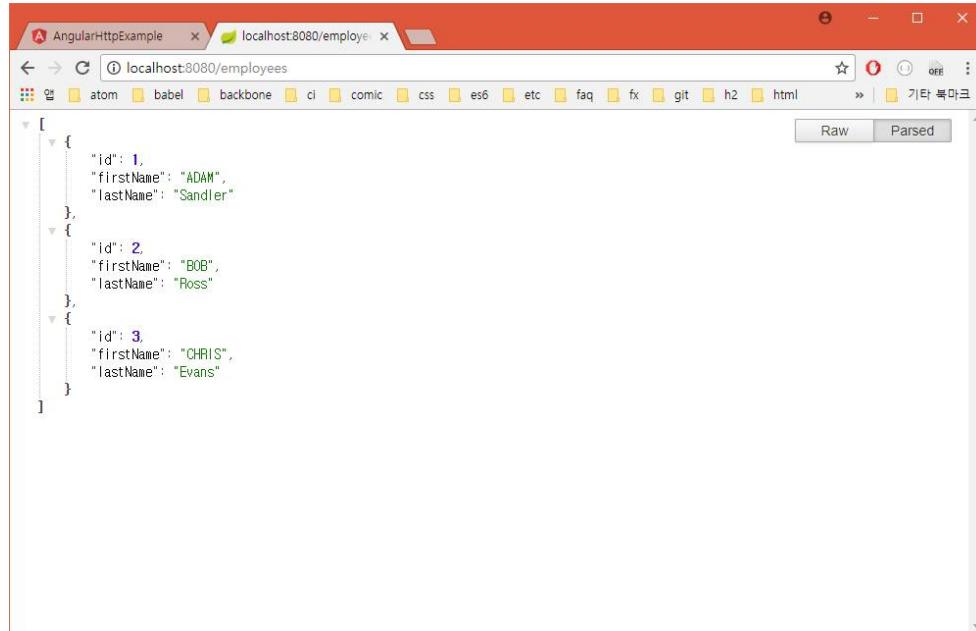
    @PostMapping("/employees")
    public Object post(@RequestBody Employee employee) {
        if (employees.size() > 0) {
            Comparator<Employee> comp = (e1, e2) -> Integer.compare(e1.getId(), e2.getId());
            Employee emp = employees.stream().max(comp).get();
            employee.setId(emp.getId() + 1);
        } else {
            employee.setId(1);
        }

        employees.add(employee);
        return employee;
    }
}
```

```
}

@RequestMapping("/employees/{id}")
public Object delete(@PathVariable int id) {
    Employee emp = employees.stream().filter(e -> e.getId() == id).findAny().orElse(null);
    if (emp != null) {
        employees.remove(emp);
    }
    return employees;
}
}
```

서버 프로젝트 기동: 프로젝트 > Run As > Spring Boot App



### 3단계 : 클라이언트 사이드 - HTTP 로직 처리

```
cd src/app/emp && echo '' > emp.model.ts && cd ../../..
```

#### emp.model.ts

```
export class Emp {  
  constructor(  
    public id: number,  
    public firstName: string,  
    public lastName: string)  
}
```

```
ng g service emp/emp-http
```

#### emp-http.service.ts

```
import { Injectable } from '@angular/core';  
import { Http, Response, Headers, RequestOptions } from '@angular/http';  
  
import { Observable } from 'rxjs/Observable';  
import 'rxjs/add/operator/map';  
import 'rxjs/add/operator/catch';  
import 'rxjs/add/observable/throw';  
  
import { Emp } from './emp.model';  
  
@Injectable()  
export class EmpHttpService {  
  private empsUrl: string = "http://localhost:8080/employees";  
  private headers = new Headers({ 'Content-Type': 'application/json; charset=utf-8' });  
  
  constructor(private http: Http) {}  
  
  getEmps(): Observable<Emp[]> {  
    return this.http.get(this.empsUrl)  
      .map(this.extractData)  
      .catch(this.handleError);  
  }  
}
```

```

addEmp(firstName: string, lastName: string): Observable<Emp> {
  let headers = new Headers({ 'Content-Type': 'application/json' });
  let options = new RequestOptions({ headers: headers });
  let emp = { "id": 0, "firstName": firstName, "lastName": lastName };

  return this.http.post(this.empsUrl, JSON.stringify(emp), options)
    .map(this.extractDataForObject)
    .catch(this.handleError);
}

removeEmp(emp): Observable<Emp[]> {
  const url = `${this.empsUrl}/${emp.id}`;
  return this.http.delete(url, { headers: this.headers })
    .map(this.extractData).catch(this.handleError);
}

private extractData(res: Response) {
  console.log('res = ' + JSON.stringify(res));
  let json = res.text();
  json = JSON.parse(json);
  return json || [];
}

private extractDataForObject(res: Response) {
  console.log('res = ' + JSON.stringify(res));
  let json = res.text();
  json = JSON.parse(json);
  return json || {};
}

private handleError(res: Response) {
  console.log(res);
  return Observable.throw(res.json().error || 'Server Down');
}
}

```

## app.module.ts

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

```

```

import { AppRoutingModule } from './app-routing.module';
import { FormsModule }   from '@angular/forms';
import { HttpClientModule }   from '@angular/http';

import { AppComponent } from './app.component';
import { KpopComponent } from './kpop/kpop.component';
import { EmpComponent } from './emp/emp.component';

import { KpopHttpService } from './kpop/kpop-http.service';
import { EmpHttpService } from './emp/emp-http.service';

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule
  ],
  declarations: [
    AppComponent,
    KpopComponent,
    EmpComponent
  ],
  providers: [KpopHttpService, EmpHttpService],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

### **emp.component.ts**

```

import { Component, OnInit } from '@angular/core';
import { EmpHttpService } from './emp-http.service';
import { Emp } from './emp.model';

@Component({
  selector: 'app-emp',
  templateUrl: './emp.component.html',
  styleUrls: ['./emp.component.css']
})
export class EmpComponent implements OnInit {
  emps: Emp[] = [
    { id: 1, firstName: 'AA', lastName: 'aa' },

```

```

{ id: 2, firstName: 'BB', lastName: 'bb' },
{ id: 3, firstName: 'CC', lastName: 'cc' },
{ id: 4, firstName: 'DD', lastName: 'dd' },
];
errorMessage: string;

constructor(private empHttpService: EmpHttpService) { }

ngOnInit() {
  this.getEmps();
}

getEmps() {
  this.empHttpService.getEmps()
    .subscribe(emps => this.emps = emps, error => this.errorMessage = <any>error);
}

addEmp(firstName: string, lastName: string) {
  this.empHttpService.addEmp(firstName, lastName)
    .subscribe(emp => this.emps.push(emp), error => this.errorMessage = <any>error);
}

removeEmp(emp: Emp) {
  this.empHttpService.removeEmp(emp)
    .subscribe(emps => this.emps = emps, error => this.errorMessage = <any>error);
  return false; // anchor 태그의 이벤트 전파를 막는다.
}

onSubmit(f) {
  if (f.valid) {
    var emp = f.value;
    this.addEmp(emp.firstName, emp.lastName);
  }
}
}

```

### emp.component.html

```

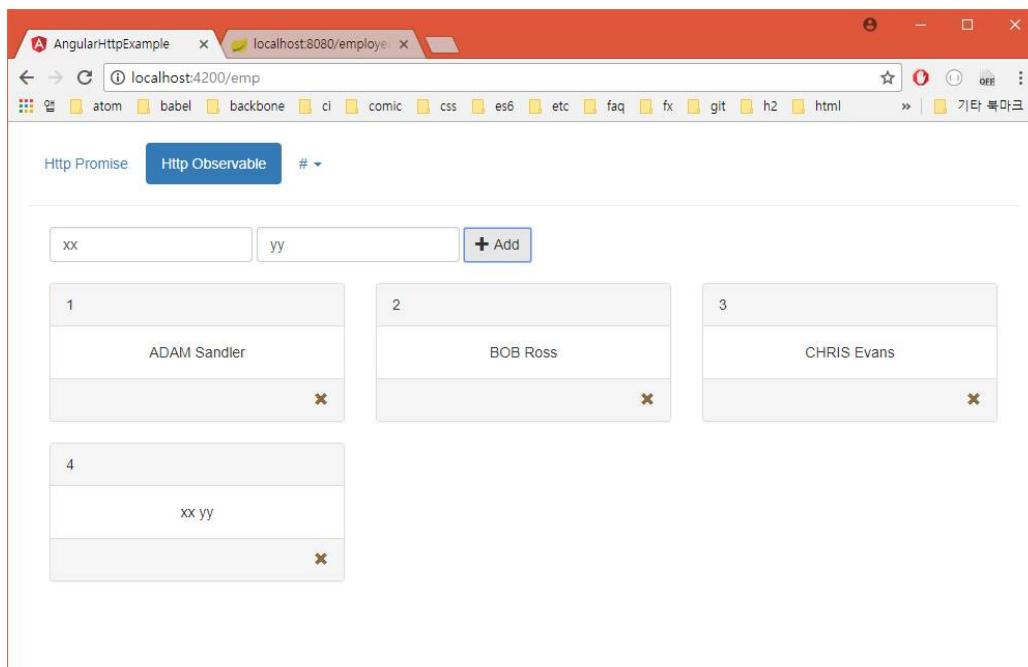
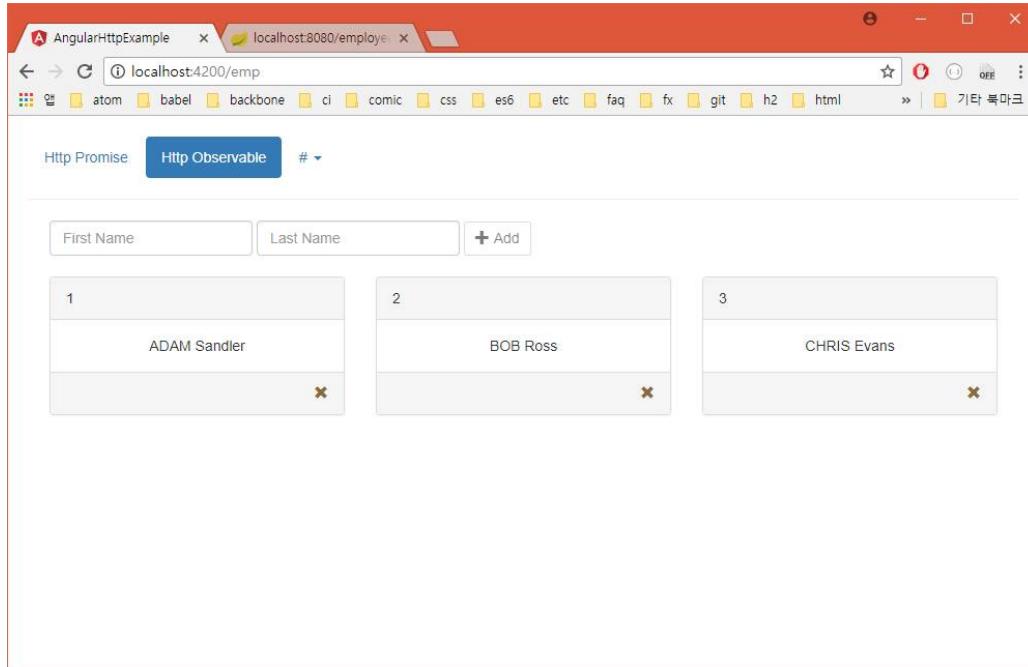
<div class="bs-example">
  <div class="panel panel-warning" *ngIf="errorMessage">
    <div class="panel-body">
      <b class="text-warning">{{errorMessage}}</b>
    
```

```

</div>
</div>
<div class="row">
<div class="col-md-12">
<form class="form-inline" #f="ngForm" (ngSubmit)="onSubmit(f)">
<div class="form-group">
<label class="sr-only" for="firstName">First Name</label>
<input type="text" class="form-control" [(ngModel)]="firstName"
id="firstName" name="firstName" placeholder="First Name" required>
</div>
<div class="form-group">
<label class="sr-only" for="lastName">Last Name</label>
<input type="text" class="form-control" [(ngModel)]="lastName"
id="lastName" name="lastName" placeholder="Last Name" required>
</div>
<!--
<button type="button" class="btn btn-default btn-sm"
[disabled]="f.invalid" (click)="addEmp(firstName, lastName)">
<b class="glyphicon glyphicon-plus"></b> Add
</button> -->
<button type="submit" class="btn btn-default btn-sm" [disabled]="f.invalid">
<b class="glyphicon glyphicon-plus"></b>
Add
</button>
</form>
</div>
</div>
<br>
<div class="row">
<div *ngFor="let emp of emps" class="col-md-4">
<div class="panel panel-default">
<div class="panel-heading">>{{emp.id}}</div>
<div class="panel-body">
{{emp.firstName}}>{{' '}}>{{emp.lastName}}
</div>
<div class="panel-footer clearfix">
<div class="pull-right">
<a href="#" class="text-warning" (click)="removeEmp(emp)">
<span class="glyphicon glyphicon-remove"></span>
</a>
</div>
</div>
</div>

```

```
</div>
</div>
</div>
</div>
```



## Step 4 – Router

라우터는 사용자가 요청한 URL에 따라 해당하는 컴포넌트를 표시하는 역할을 수행한다.

컴포넌트의 출력 영역은 <router-outlet></router-outlet>으로 정의한다.

라우터 아울렛은 루트 컴포넌트나 특징 컴포넌트에 설정한다.

루트 컴포넌트	특징 컴포넌트	자식 컴포넌트
/root	/root/children	/root/children/child1

HTML은 앵커태그의 href 속성으로 화면전환을 한다. 이 속성은 화면 전체를 로딩하기 때문에 사용하지 않고 대신 routerLink 디렉티브를 사용하여 컴포넌트 간 라우팅이 일어나게 한다.

```
<a routerLink="/root">루트 컴포넌트</a>
```

주소에 / 를 붙이면 절대주소로 취급한다. 도메인 다음부터 설정된 주소를 사용한다.

주소에 ../ 를 붙이면 상대주소로 취급한다. 현재 주소에서 한단계 위로 이동한 다음 설정된 주소를 사용한다.

클래스에서 코드적으로 라우팅 요청을 할 수 있다.

```
this._router.navigateByUrl("/root/children");

this._router.navigate(["root", "children"]);

let url = this._router.createUrlTree(["root","children"]);

this._router.navigateByUrl(url);

let url = this._router.createUrlTree([{segmentPath: "children"}]);

this._router.navigateByUrl(url);
```

## src/app/app.routing.ts

```
import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { LocationStrategy, HashLocationStrategy } from '@angular/common';

import { loginRoutes, authProviders } from './login.routing';
import { CanDeactivateGuard } from './can-deactivate-guard.service';
import { AuthGuard } from './auth-guard.service';

import { IntroComponent } from './intro.component';
import { NotFoundComponent } from './not-found.component';
import { HelloComponent } from './hello/hello.component';
import { FirstPageComponent } from './pages/first-page.component';
import { SecondPageComponent } from './pages/second-page.component';
import { ThirdPageComponent } from './pages/third-page.component';
import { RouterLinkTestComponent } from './router-link-test/router-link-test.component';
import { HrefTestComponent } from './router-link-test/href-test.component';
import { LoginComponent } from './login.component';
import { AdminComponent } from './admin/admin.component';

// Config
let hashLocationStrategy: boolean = false;

const helloRoutes: Routes = [
  { path: '', component: IntroComponent },
  { path: 'hello', component: HelloComponent },
  { path: 'router-link-test', component: RouterLinkTestComponent },
  { path: 'href-test', component: HrefTestComponent },
  { path: 'pages/first-page', component: FirstPageComponent },
  { path: 'pages/second-page', component: SecondPageComponent },
  { path: 'pages/third-page', component: ThirdPageComponent },
  { path: 'login', component: LoginComponent },
  { path: 'admin', component: AdminComponent }
];

const lazyRoutes: Routes = [
  {
    path: 'lazy',
    loadChildren: 'app/player/player.module#PlayerModule',
    canLoad: [AuthGuard]
  }
];
```

```

const appRoutes: Routes = [
  ...loginRoutes,
  ...lazyRoutes,
  ...helloRoutes,
  { path: '**', component: NotFoundComponent }
];

export const appRoutingProviders: any[] = [
  authProviders,
  CanDeactivateGuard
];

if(hashLocationStrategy){
  appRoutingProviders.push({provide: LocationStrategy, useClass: HashLocationStrategy});
}

export const AppRoutingModule: ModuleWithProviders = RouterModule.forRoot(appRoutes);

```

**import** { Routes, RouterModule } **from** '@angular/router';

Routes는 라우터 설정의 구조를 정의한 인터페이스 모듈이다.

RouterModule은 디렉티브나 컴포넌트를 포함해 모듈을 만들 때 사용하는 모듈이다.

```

const helloRoutes: Routes = [
  { path: '', component: IntroComponent },
  { path: 'hello', component: HelloComponent },
  { path: 'router-link-test', component: RouterLinkTestComponent },
  { path: 'href-test', component: HrefTestComponent },
  { path: 'pages/first-page', component: FirstPageComponent },
  { path: 'pages/second-page', component: SecondPageComponent },
  { path: 'pages/third-page', component: ThirdPageComponent },
  { path: 'login', component: LoginComponent },
  { path: 'admin', component: AdminComponent }
];

```

path가 비어 있는 경우 기본 경로 / 로 취급하여 IntroComponent를 연결한다.

```

import { loginRoutes, authProviders } from './login.routing';
import { NotFoundComponent } from './not-found.component';

const appRoutes: Routes = [
  ...loginRoutes,
  ...lazyRoutes,
  ...helloRoutes,

```

```
{ path: '**', component: NotFoundComponent }
];
```

스프레드 연산자를 사용하여 라우터 설정변수를 Routes 배열에 추가한다.

\*\* 로 404 Page Not Found 시 연동할 컴포넌트를 지정한다.

```
import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
export const AppRoutingModule: ModuleWithProviders = RouterModule.forRoot(appRoutes);
```

RouterModule.forRoot 메소드로 앱 단위의 라우터 모듈을 만든다.

AppRoutingModule을 루트 모듈에서 사용한다.

```
import { loginRoutes, authProviders } from './login.routing';
import { CanDeactivateGuard } from './can-deactivate-guard.service';

export const appRoutingProviders: any[] = [
  authProviders,
  CanDeactivateGuard
];
```

라우터 인증관 관련된 서비스를 설정한다.

## src/app/app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';

/* application router settings */
import { AppRoutingModule, appRoutingProviders } from './app.routing';

/* feature module */
import { MemberModule } from './member/member.module';
import { PlayerModule } from './player/player.module';
import { ChildrenModule } from './children/children.module';

/* global components */
import { AppComponent } from './app.component';
import { IntroComponent } from './intro.component';
```

```

import { LoginComponent } from './login.component';
import { NotFoundComponent } from './not-found.component';

import { HelloComponent } from './hello/hello.component';
import { RouterLinkTestComponent } from './router-link-test/router-link-test.component';
import { HrefTestComponent } from './router-link-test/href-test.component';

import { FirstPageComponent } from './pages/first-page.component';
import { SecondPageComponent } from './pages/second-page.component';
import { ThirdPageComponent } from './pages/third-page.component';

import { AdminComponent } from './admin/admin.component';

@NgModule({
  imports: [
    BrowserModule, CommonModule, FormsModule,
    AppRoutingModule,
    MemberModule, PlayerModule, ChildrenModule
  ],
  providers: [appRoutingProviders],
  declarations: [
    AppComponent, IntroComponent, HelloComponent,
    RouterLinkTestComponent,
    HrefTestComponent,
    FirstPageComponent, SecondPageComponent, ThirdPageComponent,
    LoginComponent,
    AdminComponent,
    NotFoundComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

기동 컴포넌트인 AppComponent가 <router-outlet></router-outlet>을 포함한다.

## 해시 기반 주소로 변경

앵귤러2는 기본적으로 앵귤러1에서 사용하던 # 해시태그를 이용하지 않는다.

그러나 때로는 해시 주소 기반으로 사용하고 싶을 수도 있다.

### app.routing.ts

```
import { LocationStrategy, HashLocationStrategy } from '@angular/common';
import { loginRoutes, authProviders } from './login.routing';
import { CanDeactivateGuard } from './can-deactivate-guard.service';

let hashLocationStrategy: boolean = true;

export const appRoutingProviders: any[] = [
  authProviders,
  CanDeactivateGuard
];

if(hashLocationStrategy){
  appRoutingProviders.push({provide: LocationStrategy, useClass: HashLocationStrategy});
}
```

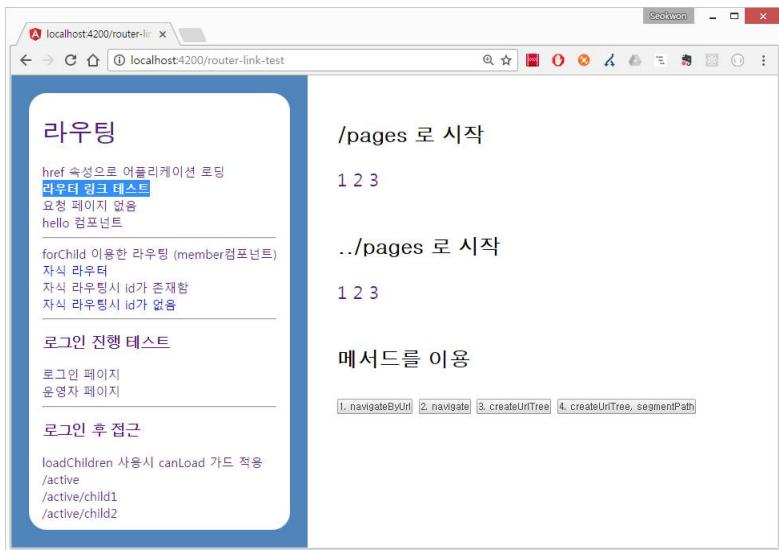
### app.module.ts

```
import { AppRoutingModule, appRoutingProviders } from './app.routing';

@NgModule({
  imports: [
    BrowserModule, CommonModule, FormsModule,
    AppRoutingModule,
    MemberModule, PlayerModule, ChildrenModule
  ],
  providers: [appRoutingProviders],
  declarations: [
    AppComponent, IntroComponent, HelloComponent,
    RouterLinkTestComponent,
    HrefTestComponent,
    FirstPageComponent, SecondPageComponent, ThirdPageComponent,
    LoginComponent,
    AdminComponent,
    NotFoundComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

http://localhost:4200/hello 주소 대신 http://localhost:4200/#/hello 주소를 사용한다.

## 연결순서 : http://localhost:4200/router-link-test



### app.component.ts

```
import {Component} from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <div class="left-menu">
      <div class="menu">
        <a routerLink="/">
          <h1>라우팅</h1>
        </a>
        <ol class="tree-list">
          <li><a routerLink="/href-test">href 속성으로 어플리케이션 로딩</a></li>
          <li><a routerLink="/router-link-test">라우터 링크 테스트</a></li>
          <li><a routerLink="/!@#">요청 페이지 없음</a></li>
          <li><a routerLink="/hello">hello 컴포넌트</a></li>
          <hr>
        </ol>
      </div>
    </div>
    <div class="play-box">
      <router-outlet></router-outlet>
    </div>
  `
})
export class AppComponent {}
```

routerLink는 URL을 라우터에게 전달한다.

라우터는 전달받은 정보에 해당하는 라우터 연결이 있으면 해당 컴포넌트를 호출한다.

## app.routing.ts

```
const helloRoutes: Routes = [
  { path: '', component: IntroComponent },
  { path: 'hello', component: HelloComponent },
  { path: 'router-link-test', component: RouterLinkTestComponent },
  { path: 'href-test', component: HrefTestComponent },
  { path: 'pages/first-page', component: FirstPageComponent },
  { path: 'pages/second-page', component: SecondPageComponent },
  { path: 'pages/third-page', component: ThirdPageComponent },
  { path: 'login', component: LoginComponent },
  { path: 'admin', component: AdminComponent }
];
```

설정에 따라 RouterLinkTestComponent 컴포넌트가 기동한다.

## router-link-test.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';

@Component({
  selector: 'router-link-test',
  template: `
    <h3>/pages 로 시작</h3>
    <a routerLink="/pages/first-page">1</a>
    <a routerLink="/pages/second-page">2</a>
    <a routerLink="/pages/third-page">3</a> <br><br>

    <h3>../pages 로 시작</h3>
    <a routerLink="../pages/first-page">1</a>
    <a routerLink="../pages/second-page">2</a>
    <a routerLink="../pages/third-page">3</a> <br><br>

    <h3>메서드를 이용</h3>
    <button (click)="one()">1. navigateByUrl</button>
    <button (click)="two()">2. navigate</button>
    <button (click)="three()">3. createUrlTree</button>
    <button (click)="four()">4. createUrlTree, segmentPath</button>
  `
})

export class RouterLinkTestComponent {
  constructor(public _router: Router) { }

  one() {
    this._router.navigateByUrl("/pages/first-page");
  }
}
```

```

}

two() {
  this._router.navigate(['pages', 'second-page']);
}

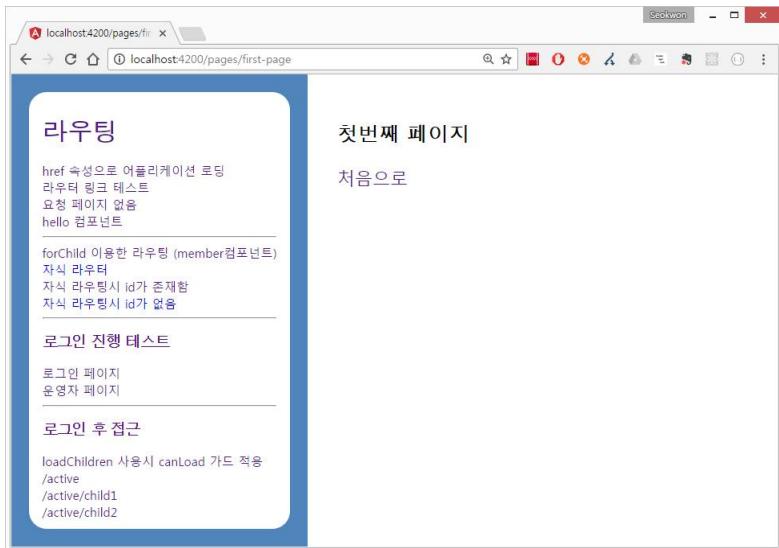
three() {
  let url = this._router.createUrlTree(['pages', 'third-page']);
  this._router.navigateByUrl(url);
}

// FIXME: bug is here
four() {
  let url = this._router.createUrlTree([{segmentPath: 'third-page'}]);
  this._router.navigateByUrl(url);
}

}

```

## 연결순서: <http://localhost:4200/pages/first-page>



## router-link-test.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';

@Component({
  selector: 'router-link-test',
  template: `
    <h3>/pages 로 시작</h3>
    <a routerLink="/pages/first-page">1</a>
    <a routerLink="/pages/second-page">2</a>
    <a routerLink="/pages/third-page">3</a> <br><br>

    <h3>..//pages 로 시작</h3>
    <a routerLink="..//pages/first-page">1</a>
    <a routerLink="..//pages/second-page">2</a>
    <a routerLink="..//pages/third-page">3</a> <br><br>

    <h3>메서드를 이용</h3>
    <button (click)="one()">1. navigateByUrl</button>
    <button (click)="two()">2. navigate</button>
    <button (click)="three()">3. createUrlTree</button>
    <button (click)="four()">4. createUrlTree, segmentPath</button>
    `

})

export class RouterLinkTestComponent {
  constructor(public _router: Router) { }

  one() {
    this._router.navigateByUrl("/pages/first-page");
  }

  two() {
    this._router.navigate(['pages', 'second-page']);
  }

  three() {
    let url = this._router.createUrlTree(['pages', 'third-page']);
    this._router.navigateByUrl(url);
  }

  // FIXME: bug is here
  four() {
    let url = this._router.createUrlTree([{segmentPath: 'third-page'}]);
  }
}
```

```
        this._router.navigateByUrl(url);
    }
}
```

## app.routing.ts

```
const helloRoutes: Routes = [
  { path: '', component: IntroComponent },
  { path: 'hello', component: HelloComponent },
  { path: 'router-link-test', component: RouterLinkTestComponent },
  { path: 'href-test', component: HrefTestComponent },
  { path: 'pages/first-page', component: FirstPageComponent },
  { path: 'pages/second-page', component: SecondPageComponent },
  { path: 'pages/third-page', component: ThirdPageComponent },
  { path: 'login', component: LoginComponent },
  { path: 'admin', component: AdminComponent }
];
```

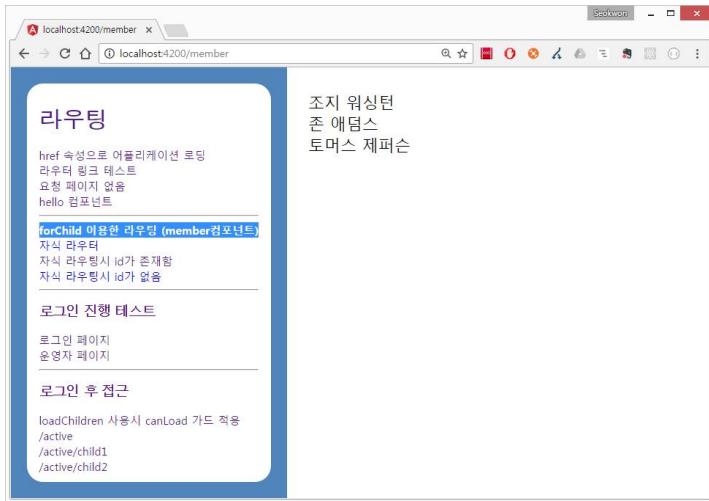
설정에 따라 FirstPageComponent 컴포넌트가 기동한다.

## first-page.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'first-page',
  template: `<h3>첫번째 페이지</h3>
  <a routerLink="/router-link-test" routerLinkActive="active">처음으로</a>`
})
export class FirstPageComponent {}
```

## 연결순서 : http://localhost:4200/member



### app.module.ts

```
/* feature module */
import {MemberModule} from './member/member.module';
import {PlayerModule} from './player/player.module';
import {ChildrenModule} from './children/children.module';

@NgModule({
  imports: [
    BrowserModule, CommonModule, FormsModule,
    AppRoutingModule,
    MemberModule, PlayerModule, ChildrenModule
  ],
  ...
  bootstrap: [AppComponent]
})
export class AppModule {}
```

### member.module.ts

```
import {NgModule}      from '@angular/core';
import {CommonModule} from '@angular/common';
import {FormsModule}  from '@angular/forms';

import {MemberComponent} from './member.component';
import {MemberRoutingModule} from './member-routing.module';

@NgModule({
  imports: [CommonModule, FormsModule, MemberRoutingModule],
  declarations: [MemberComponent],
  providers: []
})
export class MemberModule {}
```

## member-routing.module.ts

```
import {NgModule}      from '@angular/core';
import {RouterModule}  from '@angular/router';

import {MemberComponent} from './member.component';

@NgModule({
  imports: [RouterModule.forChild([
    {path: 'member', component: MemberComponent}
  ]),
  exports: [RouterModule]
})
export class MemberRoutingModule {}
```

member-routing.module.ts → member.module.ts → app.module.ts 설정으로

<http://localhost:4200/member> 주소로 접근 시 MemberComponent가 기동한다.

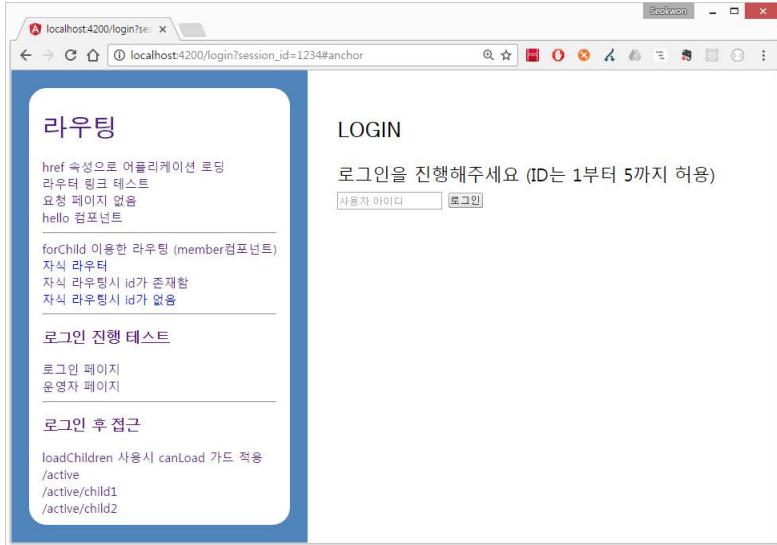
## member.component.ts

```
import {Component} from '@angular/core';

@Component({
  selector: 'member',
  template: `
    <div *ngFor='let m of member'>{{m}}</div>
  `
})
export class MemberComponent {
  member: string[] = ['조지 워싱턴', '존 애덤스', '토머스 제퍼슨'];
}
```

## 연결순서 : http://localhost:4200/children

http://localhost:4200/login?session\_id=1234#anchor 주소로 리다이렉트 된다.



### app.module.ts

```
/* feature module */
import {MemberModule} from './member/member.module';
import {PlayerModule} from './player/player.module';
import {ChildrenModule} from './children/children.module';

@NgModule({
  imports: [
    BrowserModule, CommonModule, FormsModule,
    AppRoutingModule,
    MemberModule, PlayerModule, ChildrenModule
  ],
  ...
  bootstrap: [AppComponent]
})
export class AppModule {}
```

### children.module.ts

```
import {NgModule} from '@angular/core';
//import {CommonModule} from '@angular/common';
import {FormsModule} from '@angular/forms';

import {ChildrenRoutingModule} from './children-routing.module';
import {ChildrenComponent} from './children.component';
import {Child1Component} from './child1.component';
import {Child2Component} from './child2.component';
import {Child3Component} from './child3.component';
```

```

import {ChildrenResolve} from './children-resolve.service';
import {ChildrenService} from './children.service';

@NgModule({
  imports: [ChildrenRoutingModule, FormsModule],
  declarations: [ChildrenComponent, Child1Component, Child2Component, Child3Component],
  providers: [ChildrenResolve, ChildrenService]
})
export class ChildrenModule {}

```

## children-routing.module.ts

```

import {NgModule} from '@angular/core';
import {RouterModule} from '@angular/router';

import {ChildrenComponent} from './children.component';
import {Child1Component} from './child1.component';
import {Child2Component} from './child2.component';
import {Child3Component} from './child3.component';

import {CanDeactivateGuard} from '../can-deactivate-guard.service';
import {ChildrenResolve} from './children-resolve.service';
import {AuthGuard} from '../auth-guard.service';

@NgModule({
  imports: [RouterModule.forChild([
    {
      path: 'children', component: ChildrenComponent,
      children: [{{
        path: '',
        component: Child1Component,
        children: [
          {
            path: '',
            canActivate: [AuthGuard],
            component: Child2Component
          },
          {
            path: ':id',
            component: Child3Component,
            canDeactivate: [CanDeactivateGuard],
            resolve: {
              childrenResolve: ChildrenResolve
            }
          }
        ]
      }]
    },
    {
      path: 'active', component: ChildrenComponent,
      children: [{{
        path: '',
        canActivateChild: [AuthGuard],
        children: [
          {path: 'child1', component: Child1Component},
          {path: 'child2', component: Child3Component},
          {path: '', component: Child1Component}
        ]
      }}]
    }
  ])],
  exports: [RouterModule]
})

```

```
    exports: [RouterModule]
  })
export class ChildrenRoutingModule {}
```

ChildrenComponent가 <router-outlet></router-outlet>을 갖고 있다.

<http://localhost:4200/children> 주소로 접근 시 그 영역에 Child1Component를 배치한다.

## children.component.ts

```
import {Component} from '@angular/core';

@Component({
  selector: 'children',
  template: `
    <div>자식 라우터 표시</div>
    <router-outlet></router-outlet>
  `
})
export class ChildrenComponent {}
```

Child1Component 가 <router-outlet></router-outlet>을 갖고 있다.

## child1.component.ts

```
import {Component} from '@angular/core';

@Component({
  selector: 'child1',
  template: `
    <h3>자식 1</h3>
    <router-outlet></router-outlet>
  `
})
export class Child1Component {}
```

<http://localhost:4200/children> 주소로 접근 시 그 영역에 Child2Component를 배치한다.

이 때 AuthGuard가 작동하여 접근 여부를 결정한다.

## child2.component.ts

```
import {Component} from '@angular/core';

@Component({
  selector: 'child2',
  template: `
    <h3>자식 2</h3>
  
```

```
})
export class Child2Component {}
```

http://localhost:4200/children/:id 주소로 접근 시 그 영역에 Child3Component를 배치한다.

이 때 ChildrenResolve 가드가 작동하여 라우트 데이터를 Child3Component에 제공한다.

### child3.component.ts

```
import {Component} from '@angular/core';
import {Children} from './children.service';
import {ActivatedRoute} from '@angular/router';

@Component({
  selector: 'child3',
  template: `
    <h3>자식 3</h3>
    <input type="text" [(ngModel)]="editName"> {{editName}}
  `
})
export class Child3Component {
  children: Children;
  editName: string;

  constructor(private route: ActivatedRoute) {}

  ngOnInit() {
    this.route.data.forEach((data: {childrenResolve: Children}) => {
      this.editName = data.childrenResolve.name;
      this.children = data.childrenResolve;
    });
  }
}
```

## AuthGuard

### auth-guard.service.ts

```
import {Injectable} from '@angular/core';
import {
  CanActivate, CanActivateChild, CanLoad,
  Router, NavigationExtras,
  ActivatedRouteSnapshot, RouterStateSnapshot, Route
} from '@angular/router';

import {AuthService} from './auth.service';

@Injectable()
export class AuthGuard implements CanActivate, CanActivateChild, CanLoad {
  constructor(private authService: AuthService, private router: Router) {}

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    let url: string = state.url;
    return this.checkLogin(url);
  }

  canActivateChild(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    return this.canActivate(route, state);
  }

  canLoad(route: Route): boolean {
    let url = `/${route.path}`;
    if (window.confirm("자식 라우트가 모두 로드 되었습니다. 진행하시겠습니까?")) {
      return this.checkLogin(url);
    } else {
      return false;
    }
  }

  checkLogin(url: string): boolean {
    if (this.authService.isLoggedIn) {
      return true;
    }
    this.authService.redirectUrl = url;
    let sessionId = 1234;

    let navigationExtras: NavigationExtras = {
      queryParams: {'session_id': sessionId},
      fragment: 'anchor'
    };

    this.router.navigate(['/login'], navigationExtras);
    return false;
  }
}
```

```
}
```

**this.authService.isLoggedIn**

결과가 false 이므로 다음 주소로 리다이렉트 된다.

[http://localhost:4200/login?session\\_id=1234#anchor](http://localhost:4200/login?session_id=1234#anchor)

## auth.service.ts

```
import {Injectable} from '@angular/core';
import {Observable} from 'rxjs/Observable';
import 'rxjs/add/observable/of';
import 'rxjs/add/operator/do';
import 'rxjs/add/operator/delay';

export class User {
  constructor(public id: number, public name: string) {}
}

const USERS = [
  new User(1, '첫번째 사용자'),
  new User(2, '두번째 사용자'),
  new User(3, '세번째 사용자')
];

export let userPromise = Promise.resolve(USERS);

@Injectable()
export class AuthService {
  isLoggedIn: boolean = false;
  redirectUrl: string;
  userId: string;

  checkId(userId: string): Promise<User> {
    return userPromise
      .then(children => children.find(children => children.id === +userId));
  }

  login(userId: string): Observable<boolean> {
    return Observable.of(true).delay(500)
      .do(val => this.isLoggedIn = true).do(val => this.userId = userId);
  }

  logout(): void {
    this.isLoggedIn = false;
  }
}
```

연결순서 : [http://localhost:4200/login?session\\_id=1234#anchor](http://localhost:4200/login?session_id=1234#anchor)



## app.routing.ts

```
const helloRoutes: Routes = [
  { path: '', component: IntroComponent },
  { path: 'hello', component: HelloComponent },
  { path: 'router-link-test', component: RouterLinkTestComponent },
  { path: 'href-test', component: HrefTestComponent },
  { path: 'pages/first-page', component: FirstPageComponent },
  { path: 'pages/second-page', component: SecondPageComponent },
  { path: 'pages/third-page', component: ThirdPageComponent },
  { path: 'login', component: LoginComponent },
  { path: 'admin', component: AdminComponent }
];
```

## login.component.ts

```
import {Component} from '@angular/core';
import {Router, NavigationExtras} from '@angular/router';
import {AuthService} from './auth.service';

@Component({
  template: `
    <h3>LOGIN</h3> {{message}}
    <p>
      <input type="text" [(ngModel)]="userId"
        placeholder="사용자 아이디" *ngIf="!authService.isLoggedIn">
      <button (click)="login()" *ngIf="!authService.isLoggedIn">로그인</button>
      <button (click)="logout()" *ngIf="authService.isLoggedIn">로그아웃</button>
    </p>
  `
})
export class LoginComponent {
  message: string;
  userId: string;

  constructor(public authService: AuthService, public router: Router) {
```

```

    this.setMessage();
}

setMessage() {
  this.message = (this.authService.isLoggedIn ?
    this.authService.userId + '님 로그인이 되었습니다.' :
    '로그인을 진행해주세요 (ID는 1부터 5까지 허용)');
}

private doLogin() {
  this.setMessage();
  if (this.authService.isLoggedIn) {
    let redirect = this.authService.redirectUrl ?
      this.authService.redirectUrl : '/admin';

    let navigationExtras: NavigationExtras = {
      preserveQueryParams: true,
      preserveFragment: true
    };
    this.router.navigate([redirect], navigationExtras);
  } else {
    alert('로그인을 할 수 없습니다.');
  }
}

login() {
  if (!this.userId) {
    alert('id를 입력해주세요');
    return;
  }
  this.message = '로그인을 진행해주세요';

  return this.authService.checkId(this.userId).then(children => {
    if (children) {
      this.authService.login(this.userId).subscribe(() => this.doLogin());
    } else {
      alert('아이디가 없습니다');
    }
    this.setMessage();
  });
}

logout() {
  this.authService.logout();
  this.setMessage();
}
}

```

## auth.service.ts

```

import {Injectable} from '@angular/core';
import {Observable} from 'rxjs/Observable';
import 'rxjs/add/observable/of';
import 'rxjs/add/operator/do';
import 'rxjs/add/operator/delay';

export class User {
  constructor(public id: number, public name: string) {}
}

```

```

}

const USERS = [
  new User(1, '첫번째 사용자'),
  new User(2, '두번째 사용자'),
  new User(3, '세번째 사용자')
];

export let userPromise = Promise.resolve(USERS);

@Injectable()
export class AuthService {
  isLoggedIn: boolean = false;
  redirectUrl: string;
  userId: string;

  checkId(userId: string): Promise<User> {
    return userPromise
      .then(children => children.find(children => children.id === +userId));
  }

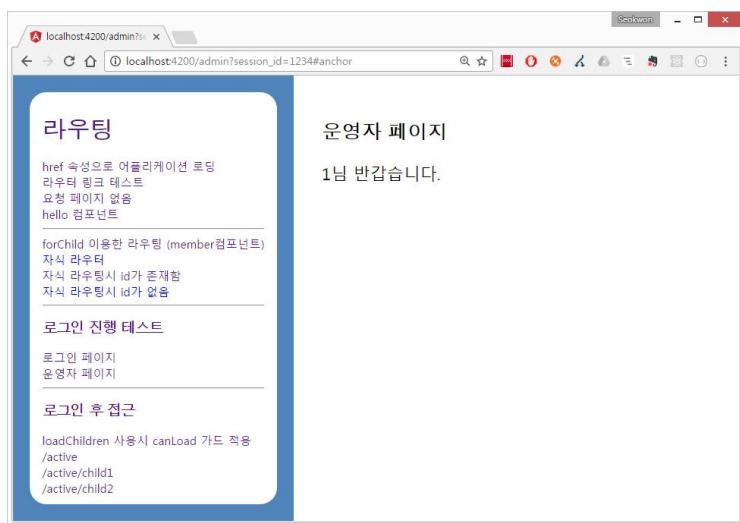
  login(userId: string): Observable<boolean> {
    return Observable.of(true).delay(500)
      .do(val => this.isLoggedIn = true).do(val => this.userId = userId);
  }

  logout(): void {
    this.isLoggedIn = false;
  }
}

```

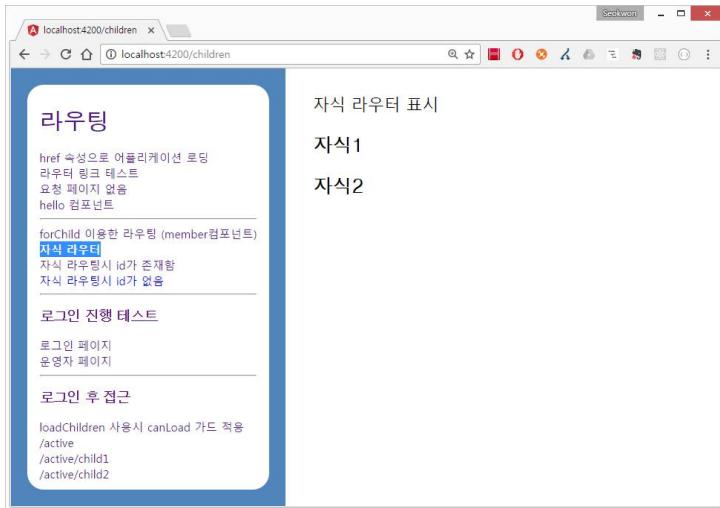
## Observable.of

Creates an Observable that emits some values you specify as arguments

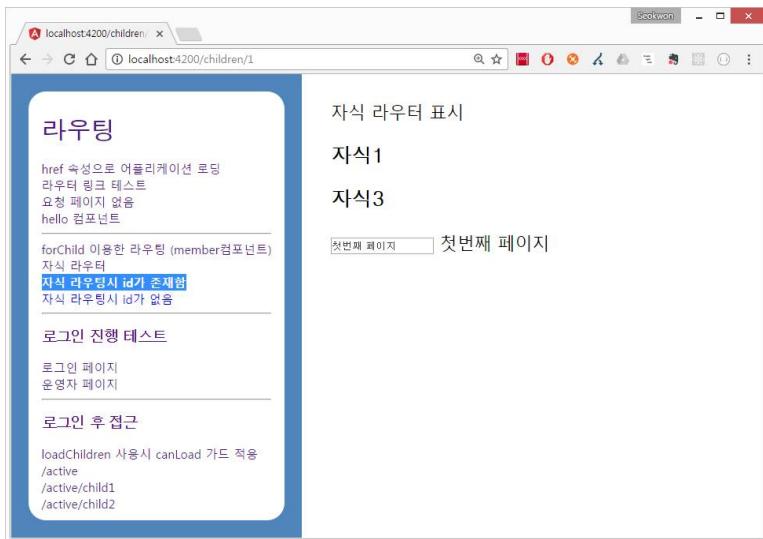


## 연결순서 : http://localhost:4200/children

로그인을 한 후 위 주소로 접속하면 원하는 화면을 볼 수 있다.



## 연결순서 : http://localhost:4200/children/1



### children-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule } from '@angular/router';
```

```

import {ChildrenComponent} from './children.component';
import {Child1Component} from './child1.component';
import {Child2Component} from './child2.component';
import {Child3Component} from './child3.component';

import {CanDeactivateGuard} from '../can-deactivate-guard.service';
import {ChildrenResolve} from './children-resolve.service';
import {AuthGuard} from '../auth-guard.service';

@NgModule({
  imports: [RouterModule.forChild([
    {
      path: 'children', component: ChildrenComponent,
      children: [
        {
          path: '',
          component: Child1Component,
          children: [
            {
              path: '',
              canActivate: [AuthGuard],
              component: Child2Component
            },
            {
              path: ':id',
              component: Child3Component,
              canDeactivate: [CanDeactivateGuard],
              resolve: {
                childrenResolve: ChildrenResolve
              }
            }
          ]
        }
      ]
    },
    {
      path: 'active', component: ChildrenComponent,
      children: [
        {
          path: '',
          canActivateChild: [AuthGuard],
          children: [
            {path: 'child1', component: Child1Component},
            {path: 'child2', component: Child3Component},
            {path: '', component: Child1Component}
          ]
        }
      ]
    }
  ])],
  exports: [RouterModule]
})
export class ChildrenRoutingModule {}

```

## children.component.ts

```

import {Component} from '@angular/core';

@Component({
  selector: 'children',
  template: `
    <div>자식 라우터 표시</div>
    <router-outlet></router-outlet>
  `
}

```

```
})
export class ChildrenComponent {}
```

## child1.component.ts

```
import {Component} from '@angular/core';

@Component({
  selector: 'child1',
  template: `
    <h3>자식 1</h3>
    <router-outlet></router-outlet>
  `
})
export class Child1Component {}
```

## children-resolve.service.ts

```
import {Injectable} from '@angular/core';
import {Router, Resolve, ActivatedRouteSnapshot} from '@angular/router';

import {Children, ChildrenService} from './children.service';

@Injectable()
export class ChildrenResolve implements Resolve<Children> {

  constructor(private cs: ChildrenService, private router: Router) {}

  resolve(route: ActivatedRouteSnapshot): Promise<Children> | boolean {
    let id = +route.params['id'];

    return this.cs.findById(id).then(children => {
      if (children) {
        return children;
      } else {
        this.router.navigate(['/not-found']);
        return false;
      }
    });
  }
}
```

## children.service.ts

```
export class Children {
  constructor(public id: number, public name: string) {}
}

const CHILDREN = [
  new Children(1, '첫번째 페이지'),
  new Children(2, '두번째 페이지'),
  new Children(3, '세번째 페이지')
];

export let childrenPromise = Promise.resolve(CHILDREN);

import {Injectable} from '@angular/core';
```

```

@Injectable()
export class ChildrenService {
  static nextId = 10;

  findById(id: number | string) {
    return childrenPromise
      .then(children => children.find(children => children.id === +id));
  }
}

```

### child3.component.ts

```

import {Component} from '@angular/core';
import {Children} from './children.service';
import {ActivatedRoute} from '@angular/router';

@Component({
  selector: 'child3',
  template: `
    <h3>자식 3</h3>
    <input type="text" [(ngModel)]="editName"> {{editName}}
  `
})
export class Child3Component {
  children: Children;
  editName: string;

  constructor(private route: ActivatedRoute) {}

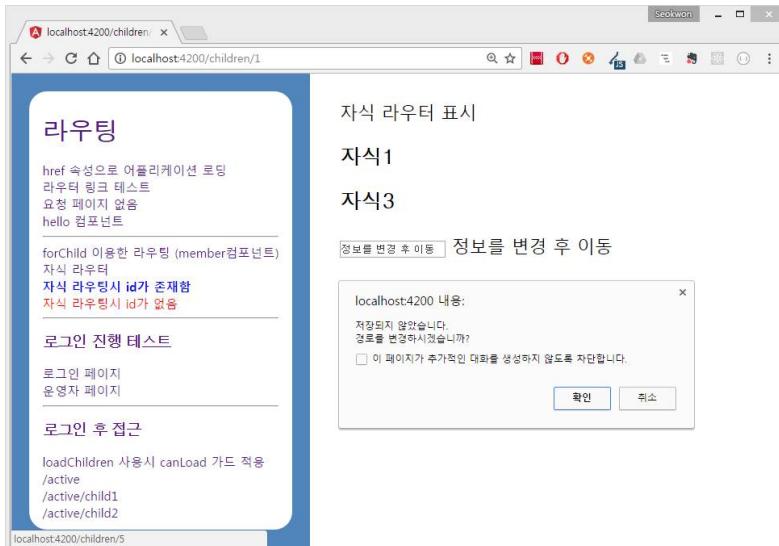
  ngOnInit() {
    console.log('this.route.data = ' + JSON.stringify(this.route.data));
    // {
    //   "_isScalar":false,
    //   "observers":[],
    //   "closed":false,
    //   "isStopped":false,
    //   "hasError":false,
    //   "thrownError":null,
    //   "_value":{
    //     "childrenResolve":{
    //       "id":1,
    //       "name":"첫번째 자식"
    //     }
    //   }
    // }

    this.route.data.forEach((data: {childrenResolve: Children}) => {
      this.editName = data.childrenResolve.name;
      this.children = data.childrenResolve;
    });
  }
}

```

## 연결순서 : /children/1 → /children/5

input 상자에 정보를 변경 후 다른 경로로 이동하려 한다.



## children-routing.module.ts

```
import {NgModule} from '@angular/core';
import {RouterModule} from '@angular/router';

import {ChildrenComponent} from './children.component';
import {Child1Component} from './child1.component';
import {Child2Component} from './child2.component';
import {Child3Component} from './child3.component';

import {CanDeactivateGuard} from '../can-deactivate-guard.service';
import {ChildrenResolve} from './children-resolve.service';
import {AuthGuard} from '../auth-guard.service';

@NgModule({
  imports: [RouterModule.forChild([
    {
      path: 'children', component: ChildrenComponent,
      children: [
        {
          path: '',
          component: Child1Component,
          children: [
            {
              path: '',
              canActivate: [AuthGuard],
              component: Child2Component
            },
            {
              path: ':id',
              component: Child3Component,
              canDeactivate: [CanDeactivateGuard],
              resolve: {
                ...
              }
            }
          ]
        }
      ]
    }
  ])
})
export class ChildrenRoutingModule {}
```

```

        childrenResolve: ChildrenResolve
    }
}
},
{
    path: 'active', component: ChildrenComponent,
    children: [
        {
            path: '',
            canActivateChild: [AuthGuard],
            children: [
                {path: 'child1', component: Child1Component},
                {path: 'child2', component: Child3Component},
                {path: '', component: Child1Component}
            ]
        }
    ],
    exports: [RouterModule]
})
export class ChildrenRoutingModule {}

```

## can-deactivate-guard.service

```

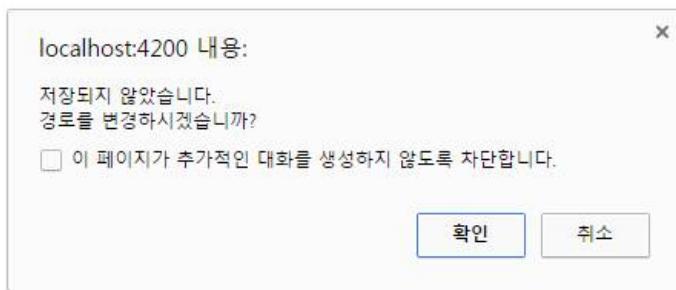
import { Injectable } from '@angular/core';
import { CanDeactivate, Router } from '@angular/router';
import { Observable } from 'rxjs/Observable';

export interface CanComponentDeactivate {
    canDeactivate: () => Observable<boolean> | Promise<boolean> | boolean;
}
@Injectable()
export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate> {

    constructor(private _router: Router) {}

    canDeactivate() {
        return window.confirm("저장되지 않았습니다.\n경로를 변경하시겠습니까?");
    }
}

```



취소를 누르면 현재 페이지가 그대로 유지된다.

확인을 누르면 <http://localhost:4200/children/5> 주소로 라우팅이 요청되지만 해당 URL에 대응하는 라우팅 정보가 없

다.

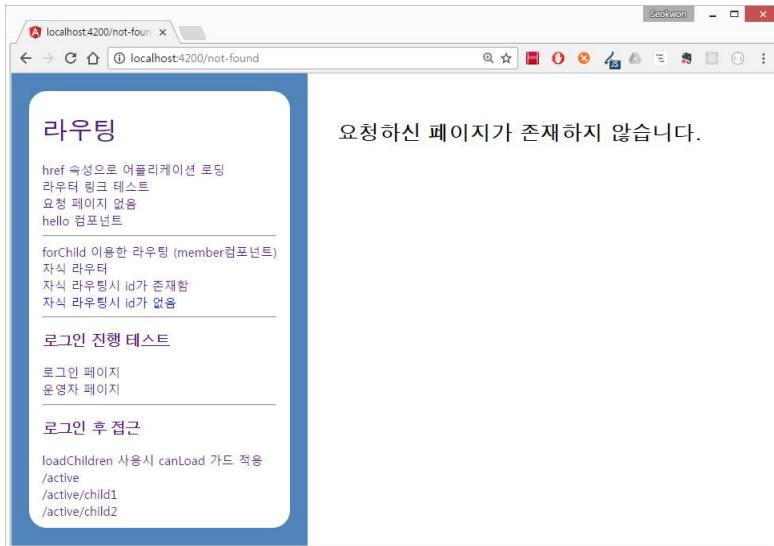
## app.routing.ts

```
const appRoutes: Routes = [
  ...loginRoutes,
  ...lazyRoutes,
  ...helloRoutes,
  { path: '**', component: NotFoundComponent }
];
```

## not-found.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'not-found',
  template: `<h3>요청하신 페이지가 존재하지 않습니다.</h3>`
})
export class NotFoundComponent {}
```



연결순서 : http://localhost:4200/lazy/player

### app.routing.ts

```
import { AuthGuard } from './auth-guard.service';

const lazyRoutes: Routes = [
{
  path: 'lazy',
  loadChildren: 'app/player/player.module#PlayerModule',
  canLoad: [AuthGuard]
}
];
```

### auth-guard.service.ts

```
import { Injectable } from '@angular/core';
import {
  CanActivate,
  Router,
  ActivatedRouteSnapshot,
  RouterStateSnapshot,
  CanActivateChild,
  NavigationExtras,
  CanLoad,
  Route
} from '@angular/router';
import { AuthService } from './auth.service';

@Injectable()
export class AuthGuard implements CanActivate, CanActivateChild, CanLoad {
  ...

  canLoad(route: Route): boolean {
    let url = `/${route.path}`;
    if (window.confirm("자식 라우트가 모두 로드 되었습니다. 진행하시겠습니까?")) {
      return this.checkLogin(url);
    } else {
      return false;
    }
  }

  checkLogin(url: string): boolean {
    if (this.authService.isLoggedIn) {
      return true;
    }
    this.authService.redirectUrl = url;
    let sessionId = 1234;

    let navigationExtras: NavigationExtras = {
      queryParams: {'session_id': sessionId},
      fragment: 'anchor'
    };

    this.router.navigate(['/login'], navigationExtras);
  }
}
```

```
        return false;
    }
}
```

로그인을 한 후 다음 처리가 진행된다.

## player.module.ts

```
import {NgModule} from '@angular/core';
import {PlayerRoutingModule} from './player-routing.module';
import {PlayerComponent} from './player.component';

@NgModule({
  imports: [PlayerRoutingModule],
  declarations: [PlayerComponent],
  providers: []
})
export class PlayerModule {}
```

## player-routing.module.ts

```
import {NgModule} from '@angular/core';
import {RouterModule} from '@angular/router';

import {PlayerComponent} from './player.component';

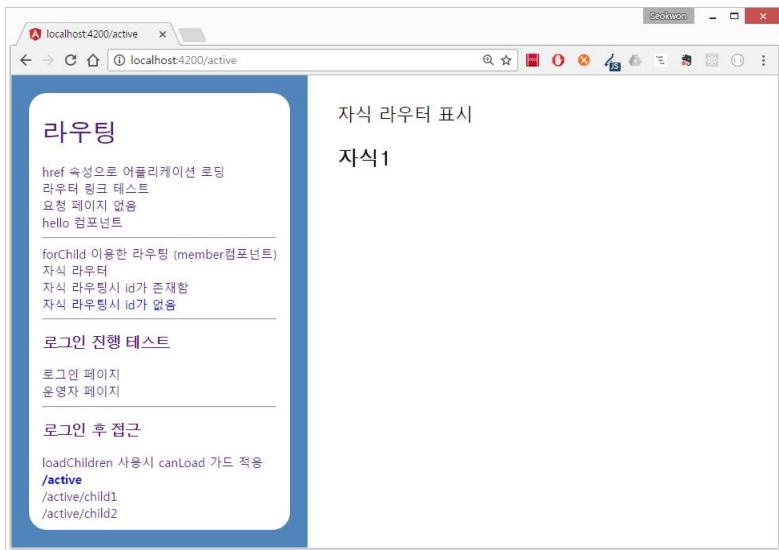
@NgModule({
  imports: [RouterModule.forChild([
    {path: '', redirectTo: '', pathMatch: 'full'},
    {path: 'player', component: PlayerComponent}
  ])],
  exports: [RouterModule]
})
export class PlayerRoutingModule {}
```

## player.component.ts

```
import {Component} from '@angular/core';

@Component({
  selector: 'player',
  template: `<h3>운영자만 볼 수 있는 Player 화면입니다.</h3>`
})
export class PlayerComponent {}
```

## 연결순서 : http://localhost:4200/active



### children-routing.module.ts

```
import {NgModule} from '@angular/core';
import {RouterModule} from '@angular/router';

import {ChildrenComponent} from './children.component';
import {Child1Component} from './child1.component';
import {Child2Component} from './child2.component';
import {Child3Component} from './child3.component';

import {CanDeactivateGuard} from '../can-deactivate-guard.service';
import {ChildrenResolve} from './children-resolve.service';
import {AuthGuard} from '../auth-guard.service';

@NgModule({
  imports: [RouterModule.forChild([
    {
      path: 'children', component: ChildrenComponent,
      children: [
        {
          path: '',
          component: Child1Component,
          children: [
            {
              path: '',
              canActivate: [AuthGuard],
              component: Child2Component
            },
            {
              path: ':id',
              component: Child3Component,
              canDeactivate: [CanDeactivateGuard],
              resolve: {
                childrenResolve: ChildrenResolve
              }
            }
          ]
        }
      ]
    }
  ])
})
export class ChildrenRoutingModule {}
```

```

        }
    ],
},
{
  path: 'active', component: ChildrenComponent,
  children: [
    {
      path: '',
      canActivateChild: [AuthGuard],
      children: [
        {path: 'child1', component: Child1Component},
        {path: 'child2', component: Child3Component},
        {path: '', component: Child1Component}
      ]
    }
  ],
}),
exports: [RouterModule]
})
export class ChildrenRoutingModule {}

```

## auth-guard.service.ts

```

import { Injectable } from '@angular/core';
import {
  CanActivate,
  Router,
  ActivatedRouteSnapshot,
  RouterStateSnapshot,
  CanActivateChild,
  NavigationExtras,
  CanLoad,
  Route
} from '@angular/router';
import { AuthService } from './auth.service';

@Injectable()
export class AuthGuard implements CanActivate, CanActivateChild, CanLoad {
  constructor(private authService: AuthService, private router: Router) {}

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    let url: string = state.url;
    return this.checkLogin(url);
  }

  canActivateChild(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    return this.canActivate(route, state);
  }

  canLoad(route: Route): boolean {

    let url = `/${route.path}`;
    if (window.confirm("자식 라우트가 모두 로드 되었습니다. 진행하시겠습니까?")) {
      return this.checkLogin(url);
    } else {
      return false;
    }
  }

  checkLogin(url: string): boolean {

```

```

if (this.authService.isLoggedIn) {
  return true;
}
this.authService.redirectUrl = url;
let sessionId = 1234;

let navigationExtras: NavigationExtras = {
  queryParams: {'session_id': sessionId},
  fragment: 'anchor'
};

this.router.navigate(['/login'], navigationExtras);
return false;
}
}

```

## child1.component.ts

```

import {Component} from '@angular/core';

@Component({
  selector: 'child1',
  template: `
    <h3>자식 1</h3>
    <router-outlet></router-outlet>`
})
export class Child1Component {}

```

<router-outlet></router-outlet>에 배정할 자식 컴포넌트를 지정하지 않았으므로 표시되지 않는다.

# 특징 모듈 라우터

□ 루트 모듈 - ♦ 특징 모듈 라우터

□ 특징 모듈 1 - ♦ 특징 모듈 라우터 1  
■ 컴포넌트 1  
■ 컴포넌트 2

□ 특징 모듈 2 - ♦ 특징 모듈 라우터 2  
■ 컴포넌트 3  
■ 컴포넌트 4

루트 모듈에서 특징 모듈의 라우터 설정 정보를 받아서 특징 모듈 간에 라우팅을 처리한다.

특징 모듈 1은 특징 모듈 라우터1을 포함하고 루트 모듈은 특징 모듈 1을 포함하는 관계이다.

예를 들어 특징 모듈 1에 있는 컴포넌트 1은 특징 모듈 2에 컴포넌트 3으로 대체될 수 있다.

## src/app/member/member-routing.module.ts

```
import { NgModule }           from '@angular/core';
import { RouterModule }        from '@angular/router';

import { MemberComponent }     from './member.component';

@NgModule({
  imports: [RouterModule.forChild([
    { path: 'member', component: MemberComponent}
  ]),
  exports: [RouterModule]
})
export class MemberRoutingModule {}
```

### RouterModule.forChild

member 특징 모듈에 대한 라우터 정보를 설정한다.

루트 모듈에 추가되기 위해서 forChild 메소드를 사용한다.

## src/app/member/member.module.ts

```
import { NgModule }           from '@angular/core';
import { CommonModule }       from '@angular/common';
import { FormsModule }        from '@angular/forms';

import { MemberComponent }    from './member.component';
import { MemberRoutingModule } from './member-routing.module';

@NgModule({
  imports:      [ CommonModule, FormsModule, MemberRoutingModule ],
  declarations: [ MemberComponent ],
  providers:    [ ]
})
export class MemberModule { }
```

## src/app/app.module.ts

```
import { BrowserModule }      from '@angular/platform-browser';
import { NgModule }            from '@angular/core';
import { CommonModule }       from '@angular/common';
import { FormsModule }         from '@angular/forms';

/* application router settings */
import { AppRoutingModule, appRoutingProviders } from './app.routing';

/* feature module */
import { MemberModule }        from './member/member.module';
import { PlayerModule }         from './player/player.module';
import { ChildrenModule }       from './children/children.module';

/* global components */
import { AppComponent }        from './app.component';
import { IntroComponent }       from './intro.component';
import { HelloComponent }       from './hello/hello.component';
import { RouterLinkTestComponent } from './router-link-test/router-link-test.component';
import { HrefTestComponent }     from './router-link-test/href-test.component';
import { FirstPageComponent }   from './pages/first-page.component';
import { SecondPageComponent }  from './pages/second-page.component';
import { ThirdPageComponent }   from './pages/third-page.component';
```

```
import { LoginComponent } from './login.component';
import { AdminComponent } from './admin/admin.component';
import { NotFoundComponent } from './not-found.component';

@NgModule({
  imports: [
    BrowserModule, CommonModule, FormsModule,
    AppRoutingModule,
    MemberModule, PlayerModule, ChildrenModule
  ],
  providers: [appRoutingProviders],
  declarations: [
    AppComponent, IntroComponent, HelloComponent,
    RouterLinkTestComponent,
    HrefTestComponent,
    FirstPageComponent, SecondPageComponent, ThirdPageComponent,
    LoginComponent,
    AdminComponent,
    NotFoundComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

## Step 5 - Guard

가드는 라우팅 시 접근을 제어하는 방법이다. 가드는 크게 다음과 같은 종류가 있다.

- CanActivate

라우트 권한을 검사해 접근 허용을 결정하는 가드이다.

- CanActivateChild

자식 라우트에 대한 접근 허용을 결정하는 가드이다.

- CanDeactivate

다른 라우터로 변경되어 현재 라우트가 비활성화될 때 호출되는 가드이다.

- Resolve

라우트 데이터를 가져와 컴포넌트에 제공하는 가드이다.

### src/app/children/children-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule } from '@angular/router';

import { ChildrenComponent } from './children.component';
import { Child1Component } from './child1.component';
import { Child2Component } from './child2.component';
import { Child3Component } from './child3.component';

import { CanDeactivateGuard } from './can-deactivate-guard.service';
import { ChildrenResolve } from './children-resolve.service';
import { AuthGuard } from './auth-guard.service';

@NgModule({
  imports: [RouterModule.forChild([
    {
      path: 'children', component: ChildrenComponent,
      children: [
        {
          path: '',
          component: Child1Component,
          children: [
            {
              path: '',
              canActivate: [AuthGuard],
            }
          ]
        }
      ]
    }
  ])
}]
```

```

        component: Child2Component
    },
    {
        path: ':id',
        component: Child3Component,
        canDeactivate: [CanDeactivateGuard],
        resolve: {
            childrenResolve: ChildrenResolve
        }
    }
]
),
{
    path: 'active', component: ChildrenComponent,
    children: [
        {
            path: '',
            canActivateChild: [AuthGuard],
            children: [
                { path: 'child1', component: Child1Component },
                { path: 'child2', component: Child3Component },
                { path: '', component: Child1Component }
            ]
        }
    ]
}),
],
exports: [RouterModule]
})
export class ChildrenRoutingModule {}
```

#### canDeactivate: [CanDeactivateGuard],

다른 컴포넌트로 변경될 때 기동한다. 예를 들어 글을 쓰다가 페이지 주소가 바뀔 때 변경사항을 저장할지 말지를 사용자에게 알리기 위한 작업을 수행할 수 있다.

```

        resolve: {
            childrenResolve: ChildrenResolve
        }
```

라우트 되는 시점에 작동해서 라우트와 관련된 데이터를 구해서 컴포넌트에게 제공하는 가드이다.

ActivateRoute 라우트를 통해 라우트되는 컴포넌트에 전달된다.

```
canActivateChild: [AuthGuard],  
children: [  
  { path: 'child1', component: Child1Component },  
  { path: 'child2', component: Child3Component },  
  { path: "", component: Child1Component }  
]
```

AuthGuard는 children 프로퍼티에 정의된 라우트 정보의 접근을 허용할지 결정한다.

```
import { ChildrenComponent } from './children.component';
```

라우터 아울렛을 통해 라우팅 요청이 들어온 컴포넌트를 표시한다.

Child1Component, Child2Component, Child3Component 컴포넌트를 사용한다.

## Step 6 – Form

### book-form-basic

#### app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule }   from '@angular/forms';
import { AppComponent }  from './app.component';
import { BookComponent } from './book.component';
import {} from '@angular/forms';

@NgModule({
  imports:      [ BrowserModule, FormsModule ],
  declarations: [ AppComponent, BookComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

#### app.componennt.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<my-book></my-book>`
})
export class AppComponent { }
```

#### book.component.ts

```
import {Component} from '@angular/core';

@Component({
  selector: 'my-book',
  templateUrl: './app/book.component.html',
  styleUrls:["./assets/stylesheets/book.css"]
})
export class BookComponent {
  onSubmit(f){
    console.log(f);
    console.log(JSON.stringify(f.form._value));
  }
}
```

```

    // {"idname": "1", "name": "2", "price": 3}
    debugger;
}
}

```

```

▼ NgForm
  _submitted: true
  control: (...)

  controls: (...)

  dirty: (...)

  disabled: (...)

  enabled: (...)

  errors: (...)

▼ form: FormGroup
  _errors: null
  ▶ _onCollectionChange: ()
  ▶ _onDisabledChange: Array[0]
  _pristine: false
  _status: "VALID"
  ▶ _statusChanges: EventEmitter
  _touched: true
  ▼ _value: Object
    date: "2015-08-23"
    idname: "<script>"
    name: "2"
    price: 4

```

## book.component.html

#변수 : 템플릿내에서 사용하는 참조

ngModel : 클래스에서 사용할 대상, 서밋 할 경우 값을 전달한다.

#f="ngForm" : 서밋 될 경우 ngForm으로 선언한 폼을 파라미터로 전달한다. ngForm 객체에서 ngModel로 선언한 값을 읽을 수 있다.

```

<h3>도서등록</h3>
<form #f="ngForm" (ngSubmit)="onSubmit(f)">
  <label for="id" style="width:100px;display:inline-block">아이디: </label>
  <input type="text" name="idname" ngModel #id required><br>
  <label for="name" style="width:100px;display:inline-block">도서명: </label>
  <input type="text" name="name" ngModel #name required><br>
  <label for="price" style="width:100px;display:inline-block">가격: </label>
  <input type="number" name="price" ngModel #price required><br>
  <label for="date" style="width:100px;display:inline-block">날짜: </label>
  <input type="date" name="date" #date><br>
  <br>
  <button type="submit">등록</button>
  <br>
  <br>
  <div style="background-color:gray">
    <div [innerText]="'bookId: ' + id.value"></div>
    <div [innerText]="'name: ' + name.value"></div>
    <div [innerText]="'price: ' + price.value"></div>
    <div [innerText]="'date: ' + date.value"></div>

```

```
</div>
</form>
```

## book-form-valid

### app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule }   from '@angular/forms';
import { AppComponent }  from './app.component';
import { BookComponent } from './book.component';

@NgModule({
  imports:      [ BrowserModule, FormsModule ],
  declarations: [ AppComponent, BookComponent ],
  bootstrap:   [ AppComponent ]
})
export class AppModule { }
```

### app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<my-book></my-book>`
})
export class AppComponent { }
```

### book.component.ts

```
import {Component} from '@angular/core';

@Component({
  selector: 'my-book',
  templateUrl: './app/book.component.html',
  styleUrls:["./assets/stylesheets/book.css"]
})
export class BookComponent {
  onSubmit(f){
    console.log(f);
  }
}
```

```

        debugger; // 디버거를 실행하면 이벤트 프로파게이션을 막는다.
    }
}

```

## book.component.html

[hidden]="id.valid" : id가 유효하면 true로 hidden 한다. \*ngIf="!id.valid" 와 같다.

처음에 작동하지 않기 위하여 보통 \*ngIf="id.touched"와 함께 사용한다.

```

<h3>도서등록</h3>
<form #f="ngForm" (ngSubmit)="onSubmit(f)">
    <label for="id" style="width:100px;display:inline-block">아이디:</label>
    <input type="text" name="id" ngModel #id="ngModel" required><br>
    <div [hidden]="id.valid" style="background-color:red;color:white">
        아이디를 입력하세요.
    </div>
    <label for="name" style="width:100px;display:inline-block">도서명:</label>
    <input type="text" name="name" ngModel #name="ngModel" required><br>
    <div [hidden]="name.valid" style="background-color:red;color:white">
        도서명을 입력하세요.
    </div>
    <label for="price" style="width:100px;display:inline-block">가격:</label>
    <input type="number" name="price" ngModel #price="ngModel" required><br>
    <div [hidden]="price.valid" style="background-color:red;color:white">
        가격을 입력하세요.
    </div>
    <label for="date" style="width:100px;display:inline-block">날짜:</label>
    <input type="date" name="date" ngModel #date="ngModel" required><br>
    <div [hidden]="date.valid" style="background-color:red;color:white">
        날짜를 입력하세요.
    </div>
    <br>
    <button type="submit">등록</button><br>
    <br>
    <div style="background-color:gray">
        <div [innerText]="'id:' + id.value"></div>
        <div [innerText]="'name:' + name.value"></div>
        <div [innerText]="'price:' + price.value"></div>
        <div [innerText]="'date:' + date.value"></div>
    </div>

```

## book-form-control

### app.module.ts

ReactiveFormsModule : 템플릿과 클래스간 엘리먼트를 공유하기 위해 필요하다.

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule, ReactiveFormsModule }   from '@angular/forms';
import { AppComponent }   from './app.component';
import { BookComponent }  from './book.component';
import {} from '@angular/forms';

@NgModule({
  imports:      [ BrowserModule, FormsModule, ReactiveFormsModule ],
  declarations: [ AppComponent, BookComponent ],
  bootstrap:   [ AppComponent]
})
export class AppModule { }
```

### book.component.ts

FormControl(초기값, 유효성 설정) : 초기값과 필수항목여부를 클래스에 정의한다.

```
import {Component} from '@angular/core';
import {FormControl, FormGroup, Validators, FormBuilder} from
'@angular/forms';

@Component({
  selector: 'my-book',
  templateUrl: './app/book.component.html',
  styleUrls:["./assets/stylesheets/book.css"]
})
export class BookComponent {
  bookForm:FormGroup = new FormGroup({
    id: new FormControl('004', Validators.required),
    name: new FormControl('초기값', Validators.required),
    price: new FormControl('', Validators.required),
    date: new FormControl('', Validators.required)
  })

  constructor(private _formBuilder:FormBuilder){
    /*
    this.bookForm= _formBuilder.group({
      id: new FormControl('111', Validators.required),
      name: new FormControl('', Validators.required),
      price:new FormControl('', Validators.required),
      date:new FormControl('', Validators.required)
    });
  }
}
```

```

    */
}

ngOnInit(){
}

onSubmit(f){
    console.log(f);
    console.log(this.bookForm);
    debugger;
}
}

```

`id: new FormControl('004', Validators.required)`

초기값으로 004를 사용한다. 필수입력항목으로 설정한다.

## book.component.html

템플릿에서도 formGroup을 추가한다. ngModel 대신 formControlName로 맵핑한다.

`*ngIf="bookForm.controls['id'].touched", [hidden]="bookForm.controls['id'].valid"` 으로 설정한다.

```

<h3>도서등록</h3>
<form [formGroup]="bookForm" #f (ngSubmit)="onSubmit(f)">
    <label for="id" style="width:100px;display:inline-block" >아이디: </label>
    <input type="text" name="id" formControlName="id" ><br>
    <div *ngIf="bookForm.controls['id'].touched"
[hidden]="bookForm.controls['id'].valid" style="background-
color:red;color:white">
        아이디를 입력하세요.
    </div>
    <label for="name" style="width:100px;display:inline-block" >도서명:</label>
    <input type="text" name="name" formControlName="name" ><br>
    <div *ngIf="bookForm.controls['name'].touched"
[hidden]="bookForm.controls['name'].valid" style="background-
color:red;color:white">
        도서명을 입력하세요.
    </div>
    <label for="price" style="width:100px;display:inline-block" >가격:</label>
    <input type="number" name="price" formControlName="price"><br>
    <div *ngIf="bookForm.controls['price'].touched"
[hidden]="bookForm.controls['price'].valid" style="background-
color:red;color:white">
        가격을 입력하세요.
    </div>
    <label for="date" style="width:100px;display:inline-block" >날짜: </label>

```

```

<input type="date" name="date" formControlName="date"><br>
<div *ngIf="bookForm.controls['date'].touched"
[hidden]="bookForm.controls['date'].valid" style="background-
color:red;color:white">
    날짜를 입력하세요.
</div>
<br>
<button type="submit">등록</button><br>
<br>
<div style="background-color:gray">
    <div [innerText]="'id:' + bookForm.controls['id'].value"></div>
    <div [innerText]="'name:' + bookForm.controls['name'].value"></div>
    <div [innerText]="'price:' + bookForm.controls['price'].value"></div>
    <div [innerText]="'date:' + bookForm.controls['date'].value"></div>
</div>
</form>

```

위 코드에서 하단 부분을 다음과 같이 바꾸고 다시 테스트 해 보자

```

<button type="submit" [disabled]="!bookForm.valid">등록</button><br>
<br>
<div style="background-color:gray">
    <div [innerText]="'id:' + bookForm.controls['id'].value "></div>
    <div [innerText]="'name:' + bookForm.controls['name'].value"></div>
    <div [innerText]="'price:' + bookForm.controls['price'].value"></div>
    <div [innerText]="'date:' + bookForm.controls['date'].value"></div>
    <div>bookForm.value : {{bookForm.value | json}}</div>
    <div>bookForm.status : {{bookForm.status}}</div>
    <div>bookForm.valid : {{bookForm.valid}}</div>
</div>

```

폼 작성 정보가 유효하지 않으면 등록 버튼이 비활성화 상태가 된다.

## book-form-formbuilder

### app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { AppComponent }   from './app.component';
import { BookComponent } from './book.component';
import {} from '@angular/forms';

@NgModule({
  imports:      [ BrowserModule, FormsModule, ReactiveFormsModule ],
  declarations: [ AppComponent, BookComponent ],
  bootstrap:    [ AppComponent]
})
export class AppModule { }
```

### app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<my-book></my-book>`
})
export class AppComponent { }
```

### book.component.ts

FormBuilder는 FormGroup과 유사하다.

```
import {Component} from '@angular/core';
import {FormControl, FormGroup, Validators, FormBuilder} from
'@angular/forms';

@Component({
  selector: 'my-book',
  templateUrl: './app/book.component.html',
  styleUrls: ["./assets/stylesheets/book.css"]
})
export class BookComponent {
  bookForm: FormGroup;

  constructor(private _formBuilder: FormBuilder) {
    this.bookForm = _formBuilder.group({
      id: new FormControl('004', Validators.required),
      title: new FormControl(''),
      author: new FormControl(''),
      year: new FormControl(''),
      price: new FormControl('')
    })
  }
}
```

```
    name: new FormControl('초기값', Validators.required),
    price: new FormControl('', Validators.required),
    date: new FormControl('', Validators.required)
  });
}

onSubmit(f) {
  console.log(f);
}
}
```

Validators.required 대신에 Validators.compose 메소드를 사용하여 조건을 복합적으로 적용할 수 있다.

- Validators.compose([Validators.required, Validators.pattern('[a-zA-Z]{3}')])
- Validators.compose([Validators.minLength(3), Validators.maxLength(5)])

# FormArray

## app.module.ts

```
import { ReactiveFormsModule } from '@angular/forms';
```

## test3.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormArray, Validators } from '@angular/forms';

@Component({
  selector: 'test3',
  template: `
    <h1>{{title}}</h1>
    <ul>
      <li *ngFor="let t of tagControls; let i=index">
        <input [formControl]="t">
        <button (click)="removeTag(i)">x</button>
      </li>
    </ul>
    <p><button (click)="addTag()">+</button></p>
    <p><button (click)="saveArticle()">Save</button></p>
  `,
  styleUrls: ['./test3.component.css']
})
export class Test3Component implements OnInit {
  title: string = 'Test3Component';

  tagControls: Array<FormControl> = [];
  tagFormArray: FormArray = new FormArray(this.tagControls);

  constructor() { }

  ngOnInit() {
  }

  addTag(): void {
    this.tagFormArray.push(new FormControl(null, Validators.required));
  }

  removeTag(idx: number): void {
    this.tagFormArray.removeAt(idx);
  }

  saveArticle(): void {
    if (this.tagFormArray.valid) {
      alert('Valid!');
    } else {
      alert('Missing field(s)!');
    }
  }
}
```



## Chapter 4 : Angular Extension

앵귤러를 제대로 사용하기 위한 핵심개념인 DI를 학습한다.

CSS를 효율적으로 적용하기 위한 설정방법을 살펴본다.

기본적으로 적용되는 HTML Escape 를 살펴보고 필요 시 이를 해제하는 사용방법을 학습한다.

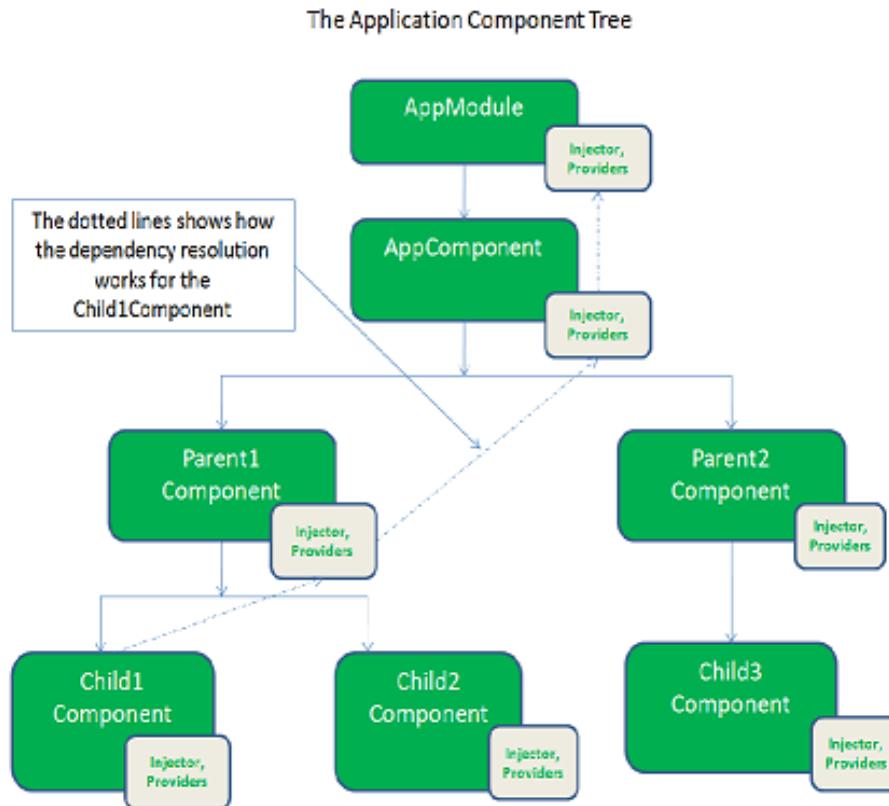
UI 의 이펙트를 주는 애니메이션 적용방법을 살펴본다.

- Dependency Injection
- CSS Style
- Sanitization
- Animation

# Step 1 – DI

<https://github.com/m00s/angular2features>

디펜던시 해결은 트리를 타고 위로 이동합니다.



```
class Car {  
  constructor(e: Engine){}  
}
```

DI는 자동차 바인딩이 정의 된 동일한 인젝터에서 엔진을 찾기 시작합니다. 해당 인젝터에 엔진 바인딩이 있는지 여부를 확인합니다. 찾는 경우 해당 인스턴스를 반환합니다. 그렇지 않은 경우, 인젝터는 부모에게 엔진 인스턴스가 있는지 여부를 묻습니다. 프로세스는 엔진의 인스턴스가 발견되거나 인젝터 트리의 루트에 도달 할 때까지 계속됩니다.

```
class Car {  
  constructor(@Self() e: Engine){}  
}
```

@Self 데코레이터는 DI에게 Car가 정의 된 동일한 인젝터에서만 엔진을 찾도록 지시합니다. 그래서 트리 위로 올라 가지 않을 것입니다.

```
class Car {  
    constructor(@Host() e: Engine){}  
}
```

@Host 데코레이터는 DI에 호스트에 도달 할 때까지 상위 인젝터에서 엔진을 찾도록 지시합니다.

```
class Car {  
    constructor(@SkipSelf() e: Engine){}  
}
```

@SkipSelf 데코레이터는 부모 인젝터에서 시작하여 전체 인젝터 트리에서 엔진을 찾도록 지시합니다.

앵귤러는 생성자 의존성 주입 패턴을 사용한다. 의존성 주입을 활용하면 객체 생성과 설정에 들어가는 코드를 최소화하고 컴포넌트마다 일관된 방법으로 객체를 사용할 수 있다.

컴포넌트 구성에 따라 injector tree가 만들어진다. 주입기 트리는 컴포넌트 구성과 일치한다. 따라서 컴포넌트마다 하나의 주입기를 갖는다. 주입기 트리에서 최상위 주입기를 root injector라고 부른다.

만약 주입 요청이 들어오고 찾는 주입 대상이 현재 주입기에 없다면 상향식으로 상위 컴포넌트에 설정한 providers 설정에서 주입 대상을 찾는다. 이러한 이유로 루트 컴포넌트에 주입 설정한 서비스의 경우 최상위 컴포넌트에 선언됐기 때문에 전역 서비스로 사용할 수 있다.

# Providers

@Injector(주입할 클래스를 선택)

→ Providers(주입할 클래스를 등록)

→ Dependency Injection(생성자로 의존성을 주입 받음)

주입할 클래스는 @Injectable 데코레이터를 사용한다. 생략이 가능하다.

변수나 함수는 @Injectable 데코레이터를 사용할 수 없다.

## 제공자의 종류

- Value Provider
- Factory Provider
- Class Provider

## 새 프로젝트 만들기

```
ng new angular-di-demo --routing=true
```

```
ng g component home
```

### index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>AngularDiDemo</title>
    <base href="/">

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
    <style type="text/css">
      .bs-example {
        margin: 20px;
      }
    </style>

  </head>
  <body>
    <app-root></app-root>
  </body>
</html>
```

### app.component.html

```
<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle"># <b class="caret"></b></a>
      <ul class="dropdown-menu">
```

```

<li><a routerLink="#">#</a></li>
<li class="divider"></li>
<li><a routerLink="#">#</a></li>
</ul>
</li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>

```

### app-routing.module.ts

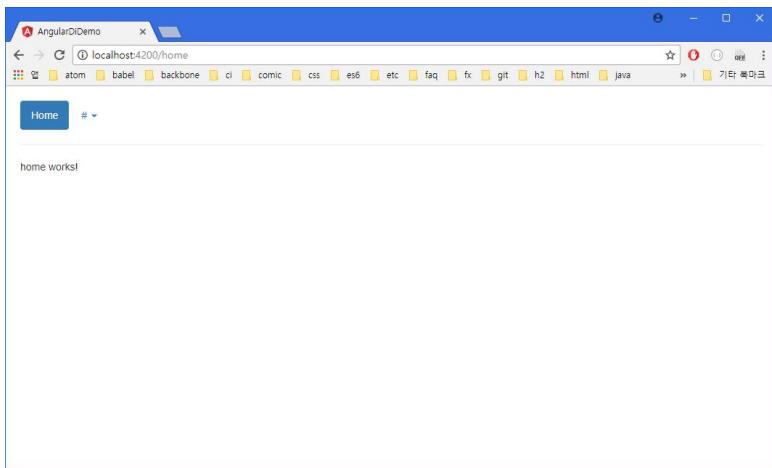
```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```



## Value Provider

```
cd src/app && mkdir provider && cd ../../
```

### src/app/provider/value.provider.ts

```
import { Injectable } from '@angular/core';

// 값 제공자에서 사용할 클래스
@Injectable()
export class Config {
    serviceName: string;
    grade: string[] = [];

    getInfo() {
    }
}

// 값 제공자에서 사용할 값
let myConfig = {
    serviceName: '회원관리 서비스',
    grade: ['운영자', '정회원', '준회원'],
    getInfo: () => {
        return `<b>${myConfig.serviceName}</b>는 <b>${myConfig.grade}</b> 등급제로 운영합니다.`
    }
};

// 값 제공자 설정
export let ValueProvider = {
    provide: Config,
    useValue: myConfig
};
```

밸류프로바이더는 Config 클래스와 이에 대한 값을 저장하고 있는 myConfig 변수를 합해 하나의 객체를 만드는 역할을 한다.

```
ng g component value-provider
```

### value-provider.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Config, ValueProvider } from './provider/value.provider';

@Component({
  selector: 'app-value-provider',
  // templateUrl: './value-provider.component.html',
  template: `
    <div>서비스명 : <b>{{config.serviceName}}</b></div>
    <div>회원등급 : <b>{{config.grade | json}}</b></div>
    <div>정보 : <b [innerHTML]="config.getInfo()"></b></div>
  `,
  styleUrls: ['./value-provider.component.css'],
  providers: [ValueProvider]
})
export class ValueProviderComponent implements OnInit {

  constructor(public config: Config) {}

  ngOnInit() {
  }

}
```

### app-routing.module.ts

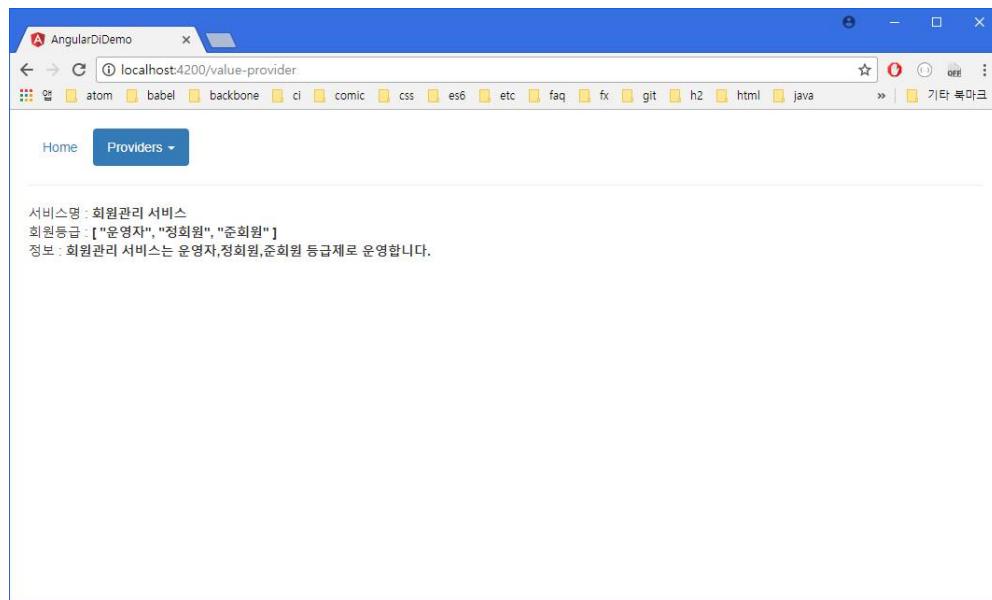
```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { ValueProviderComponent } from './value-provider/value-provider.component';

const routes: Routes = [
  { path: "", redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent},
  { path: 'value-provider', component: ValueProviderComponent},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

## app.component.html

```
<div class="bs-example">
<ul class="nav nav-pills">
<li routerLinkActive="active"><a routerLink="home">Home</a></li>
<li class="dropdown" routerLinkActive="active">
<a href="#" data-toggle="dropdown" class="dropdown-toggle">Providers <b class="caret"></b></a>
<ul class="dropdown-menu">
<li><a routerLink="value-provider">Value Provider</a></li>
<li class="divider"></li>
<li><a routerLink="#">#</a></li>
</ul>
</li>
</ul>
<hr/>
<div class="margin">
<router-outlet></router-outlet>
</div>
</div>
```



## Factory Provider

의존성 주입 시 기본적으로 싱글톤 객체가 주입된다. 때로는 싱글톤이 아닌 새롭게 생성한 객체를 의존성 주입을 통해 사용할 때가 있다.

```
ng g service provider/car
```

### car.service.ts

```
import { Injectable } from '@angular/core';

@Injectable()
export class Engine {
  public name: string;
}

@Injectable()
export class Speedmeter {
  public speed: number;
  public maxSpeed: number;
}

@Injectable()
export class CarService {
  constructor(private speedmeter: Speedmeter, public engine: Engine) { }
}
```

### src/app/provider/car.service.provider.ts

```
import { Engine, Speedmeter, CarService } from './car.service';

let carServiceFactory = (speedmeter: Speedmeter, engine: Engine) => {
  console.log('carServiceFactory() called.');
  speedmeter.speed = 100;
  speedmeter.maxSpeed = 500;
  engine.name = "슈퍼엔진";
  return new CarService(speedmeter, engine);
};

export let FactoryProvider = {
  provide: CarService,
```

```
useFactory: carServiceFactory,  
deps: [Speedmeter, Engine]  
};
```

provide: CarService

최종적으로 사용할 객체의 자료형

```
useFactory: carServiceFactory
```

팩토리 메소드

```
deps: [Speedmeter, Engine]
```

CarService 클래스가 의존하는 클래스들

```
ng g component factory-provider
```

### factory-provider.component.ts

```
import { Component, OnInit } from '@angular/core';  
import { CarService } from './provider/car.service';  
import { FactoryProvider } from './provider/car.service.provider';  
import { Speedmeter, Engine } from './provider/car.service';  
  
@Component({  
  selector: 'app-factory-provider',  
  // templateUrl: './factory-provider.component.html',  
  template: `  
    <div>엔진이름 : {{carService.engine.name}}</div>  
    <div>현재속도 : {{carService.speedmeter.speed}} km/h</div>  
    <div>최대속도 : {{carService.speedmeter.maxSpeed}} km/h</div>  
  `,  
  styleUrls: ['./factory-provider.component.css'],  
  providers: [Speedmeter, Engine, FactoryProvider]  
})  
export class FactoryProviderComponent implements OnInit {  
  
  constructor(public carService: CarService) {  
    console.log(JSON.stringify(carService));  
  }  
}
```

```

    }

    ngOnInit() {
    }

}

```

객체 설정이나 객체 생성 시 필요한 의존 객체를 컴포넌트 내부가 아닌 컴포넌트 외부에 설정했다. CarService를 주입할 때 FactoryProvider의 carServiceFactory 메소드가 새로 만들어 리턴하는 new CarService(speedmeter, engine) 객체를 사용한다.

### **app.component.html**

```

<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Providers <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="value-provider">Value Provider</a></li>
        <li><a routerLink="factory-provider">Factory Provider</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet></router-outlet>
  </div>
</div>

```

### **app-routing.module.ts**

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { ValueProviderComponent } from './value-provider/value-provider.component';
import { FactoryProviderComponent } from './factory-provider/factory-provider.component';

const routes: Routes = [

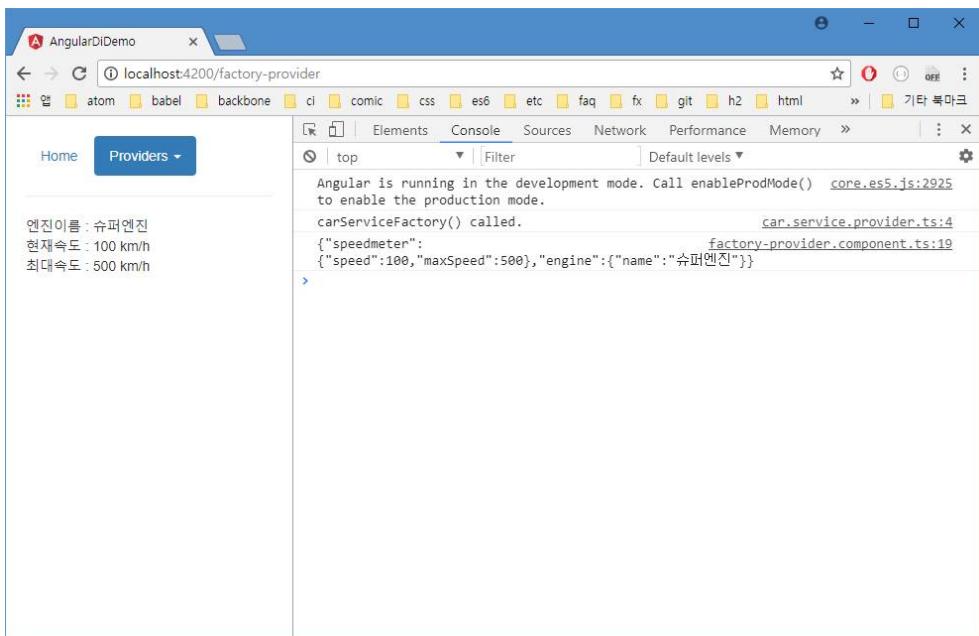
```

```

{ path: '', redirectTo: '/home', pathMatch: 'full' },
{ path: 'home', component: HomeComponent},
{ path: 'value-provider', component: ValueProviderComponent},
{ path: 'factory-provider', component: FactoryProviderComponent},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```



## Class Provider - 대체 클래스 제공자

```
ng g component class-provider
```

### engine.service.ts

```
import { Injectable } from '@angular/core';

@Injectable()
export class Engine {
  public name: string = "엔진";
}
```

### super-power-engine.service.ts

```
import { Injectable } from '@angular/core';

@Injectable()
export class SuperPowerEngine {
  public name: string = "슈퍼엔진";
  public description = '세계 최고의 성능을 자랑하는 슈퍼엔진';
}
```

### class-provider.component.ts

```
import { Component, OnInit } from '@angular/core';
import { SuperPowerEngine as spEngine } from './super-power-engine.service';
import { Engine } from './engine.service';

@Component({
  selector: 'app-class-provider',
  // templateUrl: './class-provider.component.html',
  template: `
    <div>{{engine.name}}</div>
    <div>{{spEngine.name}}</div>
    <div>{{result}}</div>
  `,
  styleUrls: ['./class-provider.component.css'],
  providers: [spEngine, { provide: Engine, useClass: spEngine }]
})
export class ClassProviderComponent implements OnInit {
```

```

result: string;

constructor(public engine: Engine, public spEngine: spEngine) {
  if (engine === spEngine) {
    this.result = "두 객체는 동일 객체입니다.";
    throw new Error('Error');
  } else {
    this.result = "두 객체는 다른 객체입니다.";
  }
}

ngOnInit() {
}
}

```

alternative class provider는 제공할 클래스를 다른 클래스로 대체하는 의존성 주입 방법을 제공한다.

**providers: [spEngine, {provide: Engine, useClass: spEngine}]**

provide는 의존성 주입 시 사용할 클래스이다. 실제 클래스의 내용은 useClass에 선언된 대체 클래스를 사용한다. providers의 첫 번째 토큰으로 spEngine을 설정했지만 대체 클래스 제공자는 이를 사용하지 않고 또 하나의 spEngine 객체를 만들어서 사용한다.

**constructor(public engine: Engine, public spEngine: spEngine)**

주입 시 engine 객체가 spEngine 객체로 대체된다.

### app-routing.module.ts

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { ValueProviderComponent } from './value-provider/value-provider.component';
import { FactoryProviderComponent } from './factory-provider/factory-provider.component';
import { ClassProviderComponent } from './class-provider/class-provider.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent},
]

```

```

{ path: 'value-provider', component: ValueProviderComponent},
{ path: 'factory-provider', component: FactoryProviderComponent},
{ path: 'class-provider', component: ClassProviderComponent},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

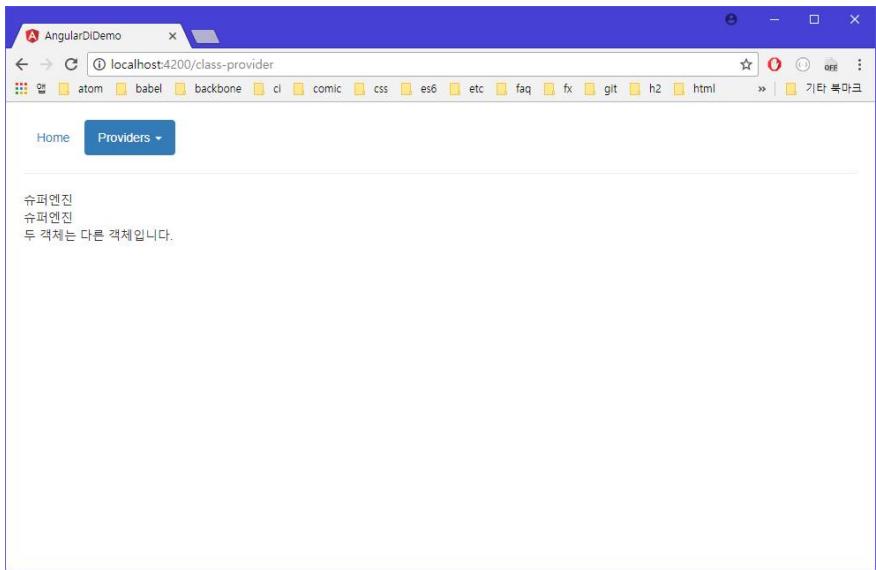
```

### app.component.html

```

<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Providers <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="value-provider">Value Provider</a></li>
        <li><a routerLink="factory-provider">Factory Provider</a></li>
        <li><a routerLink="class-provider">Class Provider</a></li>
        <li class="divider"></li>
        <li><a routerLink="#">#</a></li>
      </ul>
    </li>
  </ul>
  <hr/>
  <div class="margin">
    <router-outlet></router-outlet>
  </div>
</div>

```



## Class Provider - 가명 클래스 제공자

제공자에서 동일한 기능을 하는데 객체가 두 개 만들어지는 것은 바람직하지 않다. 싱글톤 객체를 사용하기 위해 가명 클래스 제공자를 사용한다.

class-provider.component.ts 파일을 복사하여 class-provider.component2.ts 파일을 만든다.

### class-provider.component2.ts

```
import { Component, OnInit } from '@angular/core';
import { SuperPowerEngine as spEngine } from './super-power-engine.service';
import { Engine } from './engine.service';

@Component({
  selector: 'app-class-provider',
  template: `
    <div>{{engine.name}}</div>
    <div>{{spEngine.name}}</div>
    <div>{{result}}</div>
  `,
  styleUrls: ['./class-provider.component.css'],
  // providers: [spEngine, { provide: Engine, useClass: spEngine }]
  providers: [spEngine, { provide: Engine, useExisting: spEngine }]
})
export class ClassProviderComponent2 implements OnInit {
  result: string;

  constructor(public engine: Engine, public spEngine: spEngine) {
    if (engine === spEngine) {
      this.result = "두 객체는 동일 객체입니다.";
    } else {
      this.result = "두 객체는 다른 객체입니다.";
    }
  }

  ngOnInit() {
  }
}
```

## app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { ValueProviderComponent } from './value-provider/value-provider.component';
import { FactoryProviderComponent } from './factory-provider/factory-provider.component';
import { ClassProviderComponent } from './class-provider/class-provider.component';
import { ClassProviderComponent2 } from './class-provider/class-provider.component2';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    ValueProviderComponent,
    FactoryProviderComponent,
    ClassProviderComponent,
    ClassProviderComponent2,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

## app.component.html

```
<div class="bs-example">
  <ul class="nav nav-pills">
    <li routerLinkActive="active"><a routerLink="home">Home</a></li>
    <li class="dropdown" routerLinkActive="active">
      <a href="#" data-toggle="dropdown" class="dropdown-toggle">Providers <b class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a routerLink="value-provider">Value Provider</a></li>
        <li><a routerLink="factory-provider">Factory Provider</a></li>
        <li><a routerLink="class-provider">Class Provider</a></li>
        <li><a routerLink="class-provider2">Class Provider - Alias</a></li>
```

```

<li class="divider"></li>
<li><a routerLink="#">#</a></li>
</ul>
</li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>

```

### app-routing.module.ts

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { ValueProviderComponent } from './value-provider/value-provider.component';
import { FactoryProviderComponent } from './factory-provider/factory-provider.component';
import { ClassProviderComponent } from './class-provider/class-provider.component';
import { ClassProviderComponent2 } from './class-provider/class-provider.component2';

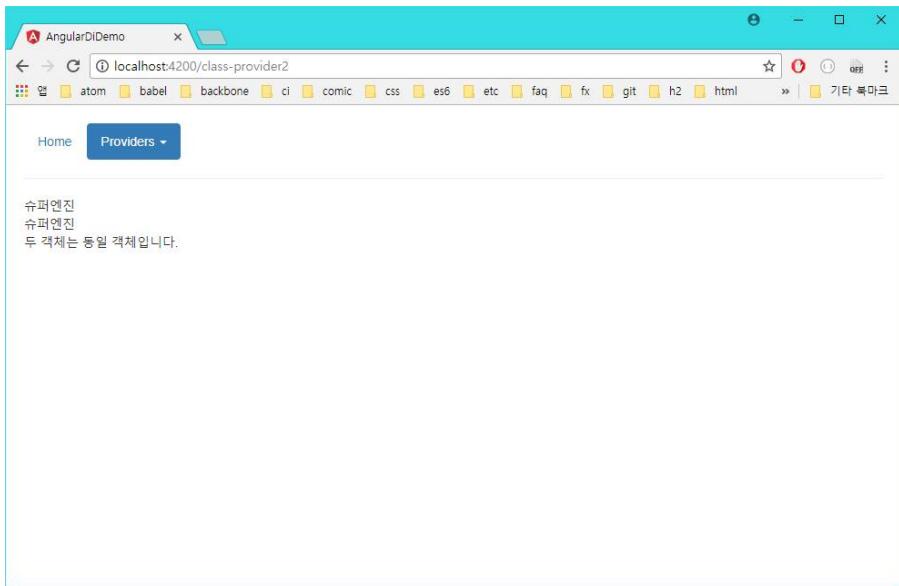
const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent},
  { path: 'value-provider', component: ValueProviderComponent},
  { path: 'factory-provider', component: FactoryProviderComponent},
  { path: 'class-provider', component: ClassProviderComponent},
  { path: 'class-provider2', component: ClassProviderComponent2},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

**providers:** [spEngine, { **provide:** Engine, **useExisting:** spEngine }]

Engine 객체가 spEngine 객체로 대체될 때 spEngine 객체가 이미 선언돼 있다면 있는 것을 그대로 사용한다.



## 불투명 토큰을 이용한 제공자 설정

opaque token은 의존성 주입 시 인터페이스를 주입받기 위해서 사용한다.

### src/app/opaque-token/opaque-token.provider.ts

```
import { Inject, OpaqueToken } from '@angular/core';

export let OPAQUE_TOKEN = new OpaqueToken('OPAQUE_TOKEN');

export interface Config {
    endpointURL: string;
    PORT: string;
}

// 인터페이스 형 변수
export const MY_API_CONFIG: Config = {
    endpointURL: 'http://192.168.0.1:80/rest',
    PORT: '8000'
};

export let OpaqueTokenProvider = {
    provide: OPAQUE_TOKEN,
    useValue: MY_API_CONFIG
};
```

타입스크립트는 의존성 주입 대상 객체에 대해 인터페이스 형을 허용하지 않는다. 인터페이스 형이어도 의존성 주입이 가능하게 하려면 provide 값을 불투명 토큰 객체로 처리해야 한다.

불투명 토큰 객체가 설정되면 useValue 값이 불투명 처리된다. 그러면 인터페이스 형일지라도 의존성 주입이 가능해진다.

### src/app/opaque-token/opaque-token.component.ts

```
import { Component, Inject } from '@angular/core';
import { MY_API_CONFIG, OpaqueTokenProvider } from './opaque-token.provider';

@Component({
    selector: 'app-inject-decorator',
    template: `
```

```
<div>API URL: {{appConfig.endpointURL}}</div>
<div>API PORT: {{appConfig.PORT}}</div>`,
providers: [{provide:OpaqueTokenProvider, useValue:MY_API_CONFIG}]
})
export class OpaqueTokenComponent {
  constructor(@Inject(OpaqueTokenProvider) public appConfig) {}
}
```

**@Inject(불투명 토큰을 설정한 제공자) public 주입 대상 변수 or 클래스**

## Provider 없이 객체 DI

외부에서 객체를 생성하고 가져다가 사용하는 방법을 factory pattern 이라고 한다.

팩토리 패턴에서는 상위 추상 클래스에서 객체의 결합 방법을 결정하고, 하위 클래스에서는 객체 설정과 생성을 수행한다.

### 팩토리 패턴 이용한 객체 주입

#### src/app/factory/animal.ts

```
import {Injectable} from '@angular/core';

export class Config {
    public bark = '어흥';
    public name = '사자';
}

@Injectable()
export class Animal {
    constructor(public config: Config) {}
    getBark() {
        return this.config.bark;
    }
    getAnimalName() {
        return this.config.name;
    }
}
```

#### src/app/factory/animal-factory.ts

```
import { Animal, Config } from './animal';

// 팩토리 패턴
abstract class Factory {
    // 객체 생성방법 결정
    create() {
        return this.createAnimal(new Config());
    }
}
```

```

abstract createAnimal(Config): Animal;
}

export class AnimalFactory extends Factory{
    // 객체 설정방법 결정
    createAnimal(config:Config){
        config.bark="야옹야옹";
        config.name="고양이";
        return new Animal(config);
    }
}

export function mainFactory() {
    let myAnimal:Animal= (new AnimalFactory()).create();
    return myAnimal;
}

```

객체 생성 방법은 Factory 추상 클래스에 정의된 create 메소드가 결정한다.

객체 설정 방법은 AnimalFactory 클래스에 정의된 createAnimal 메소드에서 결정한다.

## src/app/factory/factory.component.ts

```

import { Component } from '@angular/core';
import { mainFactory } from './animal-factory';

@Component({
    selector: 'app-factory',
    template: `
        {{animal1.getAnimalName()}} : {{animal1.getBark()}}<br>
        {{animal2.getAnimalName()}} : {{animal2.getBark()}}<br>
        {{animal1 === animal2}}
    `
})
export class FactoryComponent{
    animal1= mainFactory();
    animal2= mainFactory();
}

```

## 주입기를 이용한 객체 생성

### src/app/reflactive-injector/dog.ts

```
import {Injectable} from '@angular/core';

export class Config {
  public walking = '詈랑詈랑';

}

interface Animal {
  getName();
}

@Injectable()
export class Dog implements Animal {
  constructor(public config: Config) { }
  walking() {
    return this.config.walking;
  }
  getName() {
    return "강아지";
  }
}

@Injectable()
export class Pet extends Dog {
  constructor(public config: Config) { super(config); }

  run() {
    return this.config.walking;
  }

  getName() {
    return "애완견";
  }
}
```

## src/app/reflactive-injector/animal-injector.ts

```
import { ReflectiveInjector } from '@angular/core';
import { Dog, Pet, Config } from './dog';

export function configInjector() {
  let injector: ReflectiveInjector = ReflectiveInjector.resolveAndCreate([Config]);
  return injector.get(Config);
}

export function dogInjector() {
  let injector: ReflectiveInjector = ReflectiveInjector.resolveAndCreate([Dog, Pet, Config]);
  return injector.get(Dog);
}
```

ReflectiveInjector.**resolveAndCreate**([Config]);

토큰으로 입력된 클래스를 싱글톤 객체로 만들어주는 메소드이다.

Provider 배열을 입력받고 Injector를 리턴한다.

ReflectiveInjector.**resolveAndCreate**([Dog, Pet, Config]);

의존하는 클래스를 함께 선언해야 한다.

## src/app/reflactive-injector/animal-injector-test.ts

```
import { ReflectiveInjector } from '@angular/core';
import { Dog, Pet, Config } from './dog';

export function equalTest() {
  let injector: ReflectiveInjector = ReflectiveInjector.resolveAndCreate([Dog, Config]);
  let injector2: ReflectiveInjector = ReflectiveInjector.resolveAndCreate([Dog, Config]);
  let dog1 = injector.get(Dog);
  let dog2 = injector2.get(Dog);
  // 인젝터로부터 받은 두 객체가 동일한지 테스트한다.
  return dog1 === dog2;
}
```

```

export function equalTestNew() {
    // 수동으로 만든 두 객체가 동일한지 테스트한다.
    return (new Dog(new Config()) === new Dog(new Config()));
}

export function equalTestChild() {
    let injector: ReflectiveInjector = ReflectiveInjector.resolveAndCreate([Dog, Pet, Config]);
    let injector2: ReflectiveInjector = injector.resolveAndCreateChild([Pet]);
    let pet1 = injector.get(Pet);
    let pet2 = injector2.get(Pet);
    // 부모 인젝터로 만든 객체와 자식 인젝터로 만든 객체가 같은지 테스트한다.
    return pet1 === pet2;
}

export function fromResolvedProvidersTest() {
    let providers = ReflectiveInjector.resolve([Dog, Pet, Config]);
    // 프로바이더 정보를 활용해 새로운 인젝터를 만든다.
    let injector: ReflectiveInjector = ReflectiveInjector.fromResolvedProviders(providers);
    // 인젝터에 선언한 클래스 개수가 3개인지 테스트한다.
    return (injector.get(Pet) instanceof Pet) && providers.length == 3;
}

```

## src/app/reflection-injector/reflection-injector.component.ts

```

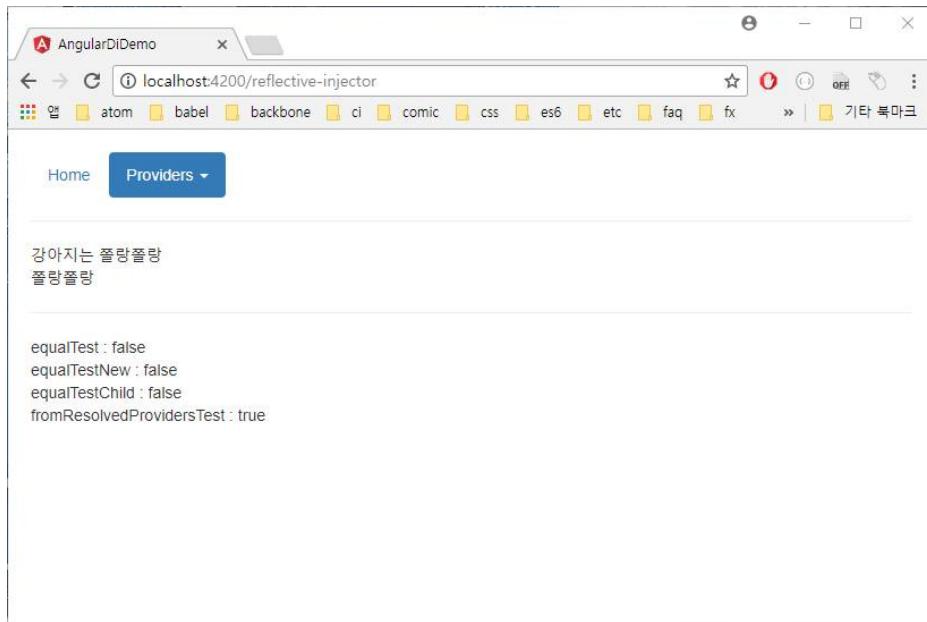
import { Component } from '@angular/core';
import { dogInjector, configInjector, } from './animal-injector';
import { equalTest, equalTestNew, equalTestChild, fromResolvedProvidersTest } from './animal-injector-test';

@Component({
    selector: 'app-reflection-injector',
    template: `
        {{dog.getName()}}는 {{dog.walking()}} <br>
        {{config.walking}}, <br>
        equalTest : {{equalTest}}<br>
        equalTestNew : {{equalTestNew}}<br>
        equalTestChild : {{equalTestChild}}<br>
        fromResolvedProvidersTest : {{fromResolvedProvidersTest}}
    `
})

```

```
export class ReflectiveInjectorComponent {
  dog = dogInjector();
  config = configInjector();

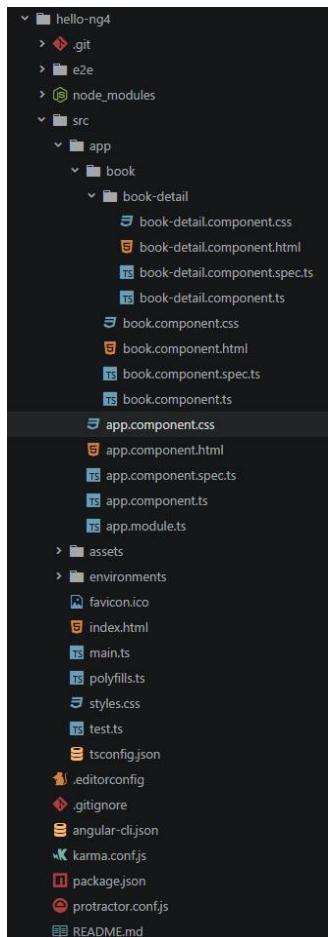
  equalTest = equalTest();
  equalTestNew = equalTestNew();
  equalTestChild = equalTestChild();
  fromResolvedProvidersTest = fromResolvedProvidersTest();
}
```



인젝터가 만드는 객체는 싱글톤이 아닌 새로운 객체다.

## Step 2 – CSS Style

다음 그림을 참고하여 프로젝트를 만든다.



### app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';

import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { BookDetailComponent } from './book/book-detail/book-detail.component';
```

```
@NgModule({
  declarations: [
    AppComponent,
    BookComponent,
    BookDetailComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

### app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app component as grand parent';
}
```

### app.component.html

```
<h1>
  {{title}}
</h1>
<div class="first">
  first div
</div>
<div class="second">
  second div
</div>
<div class="last">
  last div
</div>
```

```
</div>
<div class="child">
  <book class="active"></book>
</div>
```

### app.component.css

```
/*컴포넌트 자신, :host 생략 불가*/
:host {
  display: block;
  padding: 10px;
  background-color: grey;
}

/*컴포넌트 자신 전체에 호버 이벤트가 발생할 때 적용*/
:host(:hover) {
  font-style: italic;
}

/*태그.클래스가 일치하는 엘리먼트에 적용, :host 생략 가능*/
:host div.first {
  color: red;
  font-size: 30px;
}

/*컴포넌트 자신 + 자식 컴포넌트들에게 적용*/
:host /deep/ div.second {
  color: blue;
  font-size: 30px;
}
```

### **book.component.ts**

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {
  title: string = 'book component as parent'
  constructor() { }

  ngOnInit() { }

}
```

### **book.component.html**

```
<h1>
  {{title}}
</h1>
<div class="first">
  first div
</div>
<div class="second">
  second div
</div>
<div class="last">
  last div
</div>
<div class="child">
  <book-detail class="active"></book-detail>
</div>
```

### **book.component.css**

```
/*컴포넌트 자신, :host 생략 불가*/
:host {
  display: block;
  padding: 10px;
  background-color: orange;
```

```

}

/*
부모 HTML에서 자식 컴포넌트 셀렉터(커스텀 태그) 선언 시 클래스를 설정했다면 적용
<book class="active"></book>

:host-context(.부모가 자식 셀렉터 사용 시 붙인 클래스) .현재 컴포넌트의 클래스
*/
:host-context(.active) .last{
  color: purple;
  font-size: 30px;
}

```

### **book-detail.component.ts**

```

import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'book-detail',
  templateUrl: './book-detail.component.html',
  styleUrls: ['./book-detail.component.css']
})
export class BookDetailComponent implements OnInit {
  title: string = 'book-detail component as child'
  constructor() { }

  ngOnInit() {
  }

}

```

### **book-detail.component.html**

```

<h1>
  {{title}}
</h1>
<div class="first">
  first div
</div>
<div class="second">
  second div

```

```
</div>
<div class="last">
  last div
</div>
<div class="child">
  this is child content <button>send</button>
</div>
```

### book-detail.component.css

```
:host div {
  padding: 10px;
  background-color: yellow;
}

:host-context(.active) button {
  font-weight: bold;
  color: red;
}
```

## app component as grand parent

first div  
second div  
last div

## book component as parent

first div  
second div  
last div

## book-detail component as child

first div  
second div  
last div  
this is child content send

## Step 3 – Sanitization

새니티제이션은 잠재적인 위협요소를 제거하는 기능으로써 악의적인 의도가 있는 스크립트 유입을 막아준다. 별도 설정을 하지 않아도 템플릿에서 속성 바인딩 시 새니티제이션이 적용되어 편리하다. 그러나, 때로는 새니티제이션이 적용되지 않도록 설정을 할 필요가 있다.

### src/app/trusturl/trusturl.component.ts

```
import { Component } from '@angular/core';
import { SafeUrl, DomSanitizer } from '@angular/platform-browser';

@Component({
  selector: 'app-trusturl',
  template: `
    신뢰할 수 없는 URL : <a [href]="riskyURL">연결</a><br>
    신뢰할 수 있는 URL : <a [href]="trustURL">연결</a><br>
    신뢰할 수 없는 URL : <a [href]="'https://angular.io'">연결</a><br>
    신뢰할 수 있는 URL : <a [href]="'transToTrustURL('https://angular.io')'">연결</a>`
})
export class TrusturlComponent {

  riskyURL: String;
  trustURL: SafeUrl;

  constructor(private _sanitizer: DomSanitizer) {
    this.riskyURL = "javascript:alert('hello')";
    this.trustURL = this._sanitizer.bypassSecurityTrustUrl("javascript:alert('hello');");
  }

  transToTrustURL(url: string){
    return this._sanitizer.bypassSecurityTrustUrl(url);
  }
}
```

## src/app/trusthtml/trusthtml.component.ts

```
import { Component } from '@angular/core';
import { DomSanitizer, SafeHtml } from '@angular/platform-browser';

@Component({
  selector: 'app-trusthtml',
  template: `
    <h3>신뢰할 수 없는 HTML</h3>
    <div [innerHTML]="notTrustHtml"></div>

    <h3>신뢰할 수 있는 HTML</h3>
    <div [innerHTML]="trustHtml"></div> <br>
  `})
export class TrusthtmlComponent {

  trustHtml: SafeHtml;
  notTrustHtml: string = `<script>function hello(){ alert("헬로우 메서드 알림창!"); }</script>
<style>button{font-size:20px;padding:10px;font-style:italic;}</style>
<button onclick="hello()">메서드 호출 by onclick</button>
<button (click)="hello()">메서드 호출 by (click)</button> <br> <br>

<button onclick="javascript:alert('헬로우 알림창!')">Hello 알림창 띄우기</button>`;

  constructor(private _sanitizer: DomSanitizer) {
    this.trustHtml = this._sanitizer.bypassSecurityTrustHtml(this.notTrustHtml);
  }

  hello(){ alert('hello'); }
}
```

## src/app/trustresourceurl/trustresourceurl.component.ts

```
import { Component } from '@angular/core';
import { DomSanitizer, SafeResourceUrl } from '@angular/platform-browser';

@Component({
  selector: 'app-trustresourceurl',
  template: `<iframe width="250px" height="150px" [src]="trustResourceURL"></iframe>`
})
export class TrustresourceurlComponent {

  trustResourceURL: SafeResourceUrl;

  constructor(private _sanitizer: DomSanitizer) {
    let url="https://angular.io";
    this.trustResourceURL = this._sanitizer.bypassSecurityTrustResourceUrl(url);
  }
}
```

## src/app/truststyle/truststyle.component.ts

```
import { Component, Pipe, PipeTransform } from '@angular/core';
import { DomSanitizer, SafeStyle } from '@angular/platform-browser';

@Pipe({name: 'safe'})
export class Safe implements PipeTransform {
  constructor(private sanitizer: DomSanitizer) {
    this.sanitizer = sanitizer;
  }

  transform(style) {
    return this.sanitizer.bypassSecurityTrustStyle(style);
  }
}

@Component({
  selector: 'app-truststyle',
  template: `
    <img [style.background-image]="styleURL">
    <img [style.background-image]="styleURL | safe">
    <img [style.background-image]="trustStyle">
    <img [style.background-image]="'url(' + imageURL + ')'">
  `,
  styles: [`img{height:128px;width:128px;}`]
})
export class TruststyleComponent {

  imageURL: string = '/images/car.png';
  styleURL: string = "url('https://angular.io/resources/images/logos/angular2/angular.svg')";
  trustStyle: SafeStyle;

  constructor(private _sanitizer: DomSanitizer) {
    this.trustStyle = this._sanitizer.bypassSecurityTrustStyle(this.styleURL);
  }
}
```

<img [style.background-image]="'url(' + imageURL + ')'">>

원격이미지를 사용하기 때문에 가능하다.

# Step 4 – Animation

## 새 프로젝트 만들기

```
ng new angular-anim-example --routing=true
```

```
ng g component home
```

### index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>AngularAnimExample</title>
    <base href="/">

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
    <style type="text/css">
      .bs-example {
        margin: 20px;
      }
    </style>

  </head>
  <body>
    <app-root></app-root>
  </body>
</html>
```

### app.component.html

```
<div class="bs-example">
  <ul class="nav nav-pills">
```

```

<li routerLinkActive="active"><a routerLink="home">Home</a></li>
<li class="dropdown" routerLinkActive="active">
  <a href="#" data-toggle="dropdown" class="dropdown-toggle"># <b class="caret"></b></a>
  <ul class="dropdown-menu">
    <li><a routerLink="#">#</a></li>
    <li class="divider"></li>
    <li><a routerLink="#">#</a></li>
  </ul>
</li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>

```

### app-routing.module.ts

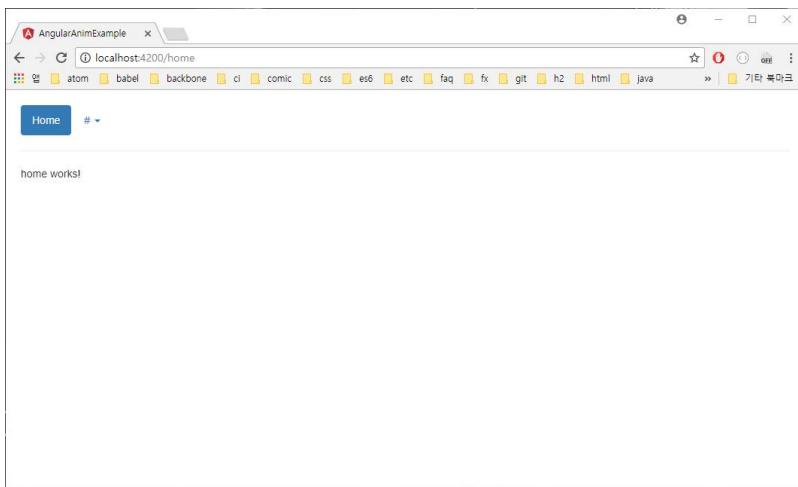
```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```



## 예제 1

### app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';

@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    BrowserAnimationsModule,
    AppRoutingModule,
  ],
  declarations: [
    AppComponent,
    HomeComponent
  ],
  providers: []
})
```

```
bootstrap: [AppComponent]
})
export class AppModule { }
```

## home.component.ts

```
import { Component, OnInit } from '@angular/core';
import { trigger, state, style, transition, animate, keyframes } from '@angular/animations';

@Component({
  selector: 'app-home',
  // templateUrl: './home.component.html',
  template: `
    <p [@myAwesomeAnimation]='state' (click)="animateMe()">I will animate</p>
  `,
  styleUrls: ['./home.component.css'],
  animations: [
    trigger('myAwesomeAnimation', [
      state('small', style({
        transform: 'scale(1)',
      })),
      state('large', style({
        transform: 'scale(1.2)',
      })),
      transition('small <=> large', animate('100ms ease-in')),
    ]),
  ]
})
export class HomeComponent implements OnInit {
  state: string = 'small';

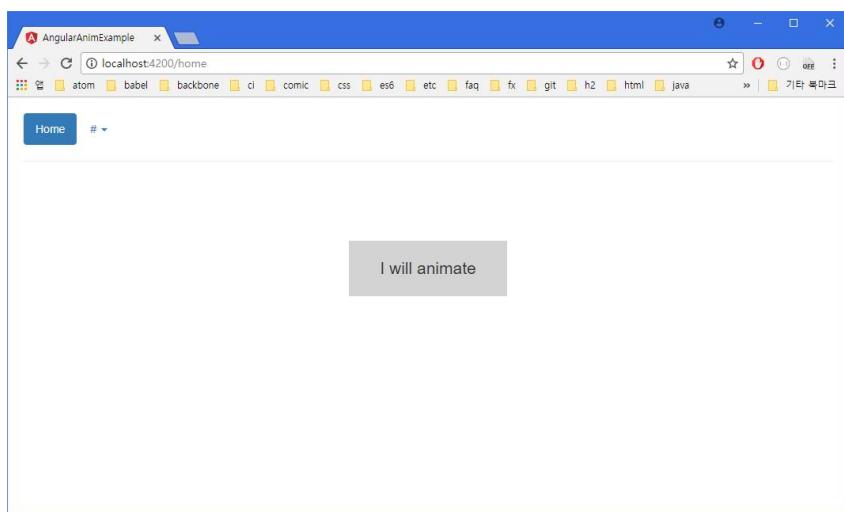
  constructor() {}

  ngOnInit() {}

  animateMe() {
    this.state = (this.state === 'small' ? 'large' : 'small');
  }
}
```

## home.component.css

```
p {  
    width: 200px;  
    background: lightgray;  
    margin: 100px auto;  
    text-align: center;  
    padding: 20px;  
    font-size: 1.5em;  
}
```



## 예제 2

```
ng g component anim1
```

## anim1.component.ts

```
import { Component, OnInit } from '@angular/core';  
import { trigger, state, style, transition, animate, keyframes } from '@angular/animations';  
  
@Component({  
    selector: 'app-anim1',  
    template: `  
        <p [@myAwesomeAnimation]='state' (click)="animateMe()">I will animate</p>
```

```

';
styleUrls: ['./anim1.component.css'],
animations: [
  trigger('myAwesomeAnimation', [
    state('small', style({
      transform: 'scale(1)',
    })),
    state('large', style({
      transform: 'scale(1.2)',
    })),
    transition('small <=> large', animate('300ms ease-in', keyframes([
      style({ opacity: 0, transform: 'translateY(-75%)', offset: 0 }),
      style({ opacity: 1, transform: 'translateY(35px)', offset: 0.5 }),
      style({ opacity: 1, transform: 'translateY(0)', offset: 1.0 })
    ]))),
  ]),
]
})
export class Anim1Component implements OnInit {
  state: string = 'small';

  constructor() { }

  ngOnInit() {
  }

  animateMe() {
    this.state = (this.state === 'small' ? 'large' : 'small');
  }
}

```

### **anim1.component.css**

```

p {
  width: 200px;
  background: lightgray;
  margin: 100px auto;
  text-align: center;
  padding: 20px;
  font-size: 1.5em;
}

```

## app-routing.module.ts

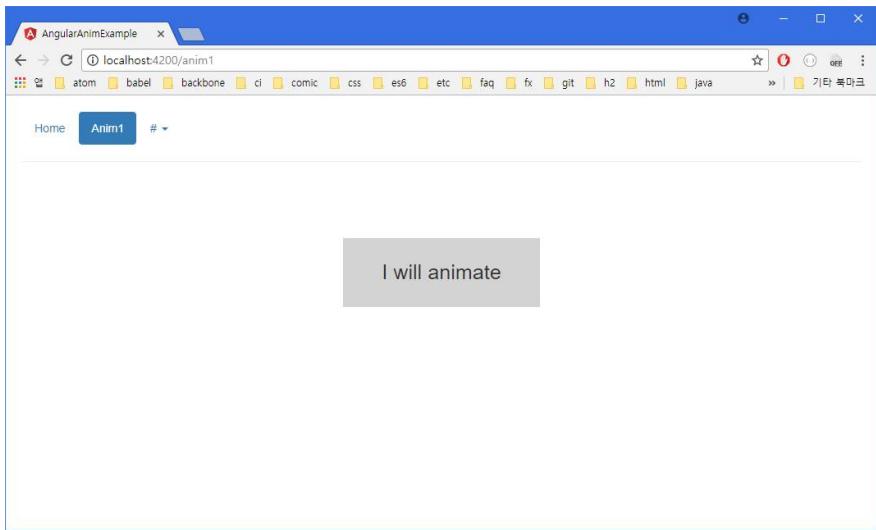
```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { Anim1Component } from './anim1/anim1.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent},
  { path: 'anim1', component: Anim1Component},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

## app.component.html

```
<div class="bs-example">
<ul class="nav nav-pills">
  <li routerLinkActive="active"><a routerLink="home">Home</a></li>
  <li routerLinkActive="active"><a routerLink="anim1">Anim1</a></li>
  <li class="dropdown" routerLinkActive="active">
    <a href="#" data-toggle="dropdown" class="dropdown-toggle"># <b class="caret"></b></a>
    <ul class="dropdown-menu">
      <li><a routerLink="#">#</a></li>
      <li class="divider"></li>
      <li><a routerLink="#">#</a></li>
    </ul>
  </li>
</ul>
<hr/>
<div class="margin">
  <router-outlet></router-outlet>
</div>
</div>
```



### 예제 3

#### anim2.component.ts

```
import { Component, OnInit } from '@angular/core';
import { animate, state, style, transition, trigger } from '@angular/animations';

@Component({
  selector: 'app-anim2',
  // templateUrl: './anim2.component.html',
  template: `
    <button (click)="expand()">Open</button>
    <button (click)="collapse()">Closed</button>
    <hr />
    <div class="toggle-container" [@openClose]="stateExpression">
      Look at this box
    </div>
  `,
  // styleUrls: ['./anim2.component.css']
  styles: [
    .toggle-container {
      background-color:white;
      border:10px solid black;
      width:200px;
      text-align:center;
    }
  ]
})
```

```

        line-height:100px;
        font-size:50px;
        box-sizing:border-box;
        overflow:hidden;
    }
],
animations: [
    trigger('openClose', [
        state('collapsed', void', style({
            height: '0px', color: 'maroon', borderColor: 'maroon'
        })),
        state('expanded', style({
            height: '*', borderColor: 'green', color: 'green'
        })),
        transition('collapsed <=> expanded', [
            animate(500, style({
                height: '250px'
            })),
            animate(500)
        ])
    ])
]
})
export class Anim2Component implements OnInit {
    stateExpression: string;

    constructor() {
        this.collapse();
    }

    ngOnInit() {
    }

    expand() {
        this.stateExpression = 'expanded';
    }

    collapse() {
        this.stateExpression = 'collapsed';
    }
}

```

### **app-routing.module.ts**

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { Anim1Component } from './anim1/anim1.component';
import { Anim2Component } from './anim2/anim2.component';

const routes: Routes = [
  { path: "", redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent},
  { path: 'anim1', component: Anim1Component},
  { path: 'anim2', component: Anim2Component},
];

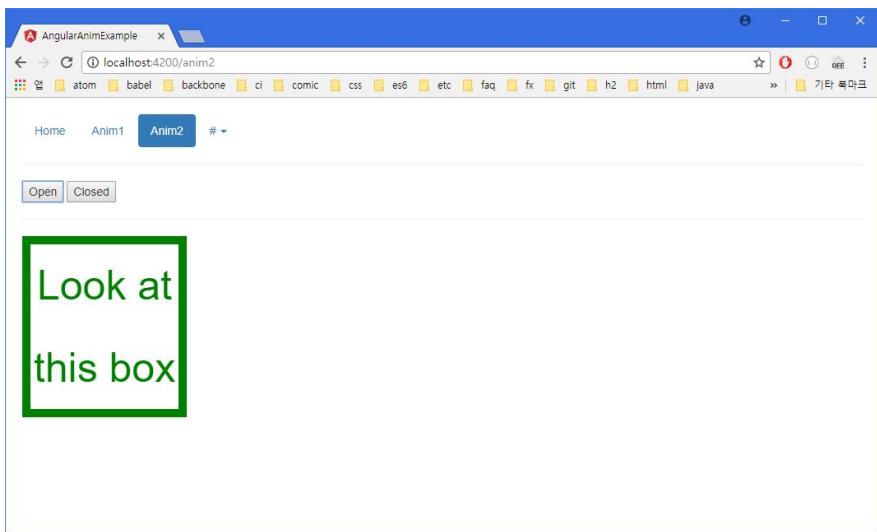
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

### **app.component.html**

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { Anim1Component } from './anim1/anim1.component';
import { Anim2Component } from './anim2/anim2.component';

const routes: Routes = [
  { path: "", redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent},
  { path: 'anim1', component: Anim1Component},
  { path: 'anim2', component: Anim2Component},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```





## Chapter 5 : Angular Deep Dive

앵귤러 1과의 차이점을 살펴본다.

실무적인 예제 프로젝트를 살펴본다.

앵귤러 공식 사이트의 튜토리얼을 직접 따라하면서 앞에서 배운 내용을 복습한다.

- Angular 1 과의 차이점
- 살펴보기 : Registration and Login
- 실습 : Tour of Heroes Tutorial

# Step 1 - Angular1과의 차이점

Angular는 Angular JS 1을 근본적으로 다시 작성한 것으로, 많은 고유한 특성을 가지고 있습니다.

Angular는 스코프 또는 컨트롤러 개념이 없으며 대신 주요 아키텍처 개념으로 컴포넌트 계층 구조를 사용합니다.

Angular는 속성 바인딩을 위한 "`[]`" 과 이벤트 바인딩을 위한 "`()`"에 초점을 맞춘 표현식을 사용합니다.

최근 프로젝트는 개발 시, 모바일 이슈를 먼저 고려하는 것이 훨씬 유리합니다.

많은 핵심 기능이 별도의 모듈에 담겨 보다 가볍고 빠르게 핵심모듈을 사용할 수 있습니다.

브라우저 호환성 해결을 위한 작업이 감소합니다.

Angular는 Microsoft의 TypeScript 언어를 사용하는 것이 좋습니다. TypeScript는 다음과 같은 기능을 제공합니다.

- 클래스 기반 객체 지향 프로그래밍
- 정적 타이핑
- 제네릭

TypeScript는 ECMAScript 6 (ES6)의 상위 집합입니다. Angular에는 ES6의 이점도 포함됩니다.

- 람다
- 이터레이터
- for-of 루프
- 파이썬 스타일의 제너레이터
- 리플렉션
- 향상된 DI - 바인딩을 사용하면 종속성을 명명 할 수 있습니다.
- 동적로딩
- 비동기식 템플릿 컴파일
- 보다 단순한 라우팅
- 컨트롤러와 \$scope 를 컴포넌트와 디렉티브로 대체 - 컴포넌트는 템플릿이 있는 디렉티브입니다
- RxJS 를 사용한 리액티브 프로그래밍 지원

## 명칭

원래, AngularJS의 재 작성은 팀에서 "Angular 2"라고 명명했지만 이는 개발자들 사이에 혼란을 야기했습니다. 명확히하기 위해 팀은 각 프레임 워크마다 "AngularJS" (1.X 버전 참조) 및 "Angular"(JS 및 버전 없음) 버전 2 이상을 사용하는 별도의 용어를 사용해야한다고 발표했습니다.

## Version 2.0.0

Angular 2.0은 ng-Europe 회의 22-23에서 발표되었습니다. 2014년 9월 2.0 버전의 급격한 변화는 개발자들 사이에 상당한 논란을 불러 일으켰다. 2015년 4월 30일 Angular 개발자는 Angular 2가 Alpha에서 Developer Preview로 이전되었음을 발표했습니다. Angular 2는 공식 웹 사이트에서 다운로드 할 수 있습니다. 앵귤러 2는 2015년 12월에 베타 버전으로 이전되었으며 첫 번째 출시 후보는 2016년 5월에 게시되었습니다. 최종 버전은 2016년 9월 14일에 출시되었습니다.

## Version 4.0.0

2016년 12월 13일 Angular 4가 발표되었는데, 이미 v3.3.0으로 배포 된 라우터 패키지 버전의 오정렬로 인한 혼란을 피하기 위해 3을 건너 뛰었습니다. 최종 버전은 2017년 3월 23일에 발표되었습니다. 앵귤러 4는 앵귤러 2와 역 호환됩니다.

## Controllers

AngularJS 애플리케이션을 구현하는 가장 좋은 방법은 컨트롤러가 DOM을 조작해서는 안되며 모든 DOM 액세스 및 조작을 지시문에서 분리해야한다는 것입니다. 컨트롤러간에 반복적인 로직이 있다면, 이를 서비스로 캡슐화하고 해당 기능이 필요한 모든 컨트롤러에 AngularJS의 의존성 주입 메커니즘을 주입하고자 할 가능성이 큽니다.

**This is where we're coming from in AngularJS 1.x.** All this said, it seems that the functionality of controllers could be moved into the directive's controllers. Since directives support the dependency injection API, after receiving the user's input, we can directly delegate the execution to a specific service, already injected. This is the main reason **Angular 2 uses a different approach by removing the ability to put controllers everywhere by using the ng-controller directive.**

앵귤러2에서는 컨트롤러가 사라졌다. 이를 컴포넌트의 클래스가 대신한다.

```
@Component({  
  selector: 'hello-world',
```

```

template: '< h1 > Hello, {{ target}}! </ h1 >' }
class HelloWorld {
  target: string;
  constructor() { this.target = 'world'; }
}

```

## Scope

The **data-binding in AngularJS is achieved using the scope object**. We can **attach properties to it(스코프)** and explicitly declare in the template that we want to bind to these properties (one or two-way). Although the idea of the scope seems clear, **the scope has two more responsibilities, including event dispatching and the change detection-related behavior**. Angular beginners have a hard time understanding what scope really is and how it should be used. AngularJS 1.2 introduced something called **controller as syntax**. It allows us **to add properties to the current context inside the given controller (this)**, instead of explicitly injecting the scope object and later adding properties to it. This simplified syntax can be demonstrated from the following snippet:

Angular 2는 아예 스코프 객체를 제거하였습니다. 모든 표현식은 주어진 UI 구성 요소의 컨텍스트에서 평가됩니다. 전체 범위 API를 제거하면 더 단순 해집니다. 명시 적으로 더 이상 삽입 할 필요가 없으며 나중에 바인딩 할 수 있는 UI 구성 요소에 속성을 추가합니다. 이 API는 훨씬 간단하고 자연 스럽습니다.

앵귤러2에서는 스코프가 사라졌다. 컴포넌트 클래스 별로 자원을 관리한다. 컴포넌트들끼리의 자원공유는 서비스를 사용한다.

## Injection

어쩌면 JavaScript 세계에서 Dependency Injection (DI)을 통한 Inversion of Control (IoC)가 포함 된 시장의 첫 번째 프레임 워크가 AngularJS 1이었을 것입니다. DI는 테스트 용이성, 코드 구성 및 모듈화 개선, 단순성과 같은 많은 이점을 제공합니다. 1.x의 DI는 놀랄만 한 일을 하지만, Angular 2는 이것을 훨씬 더 많이 사용합니다. Angular 2는 최신 웹 표준의 최상위에 있기 때문에 DI를 사용하는 코드에 메타데이터를 추가하기 위해서 ECMAScript 2016 (ES7) 데코레이터 구문을 사용합니다. Decorator는 Python의 데코레이터 또는 Java의 주석과 매우 유사합니다. 그것들은 리플렉션을 사용하여 주어진 객체의 동작을 꾸밀 수 있게 해줍니다. 데코레이터는 아직 주요 브라우저에서 표준화되지 않고 지원되지 않기 때문에 중간 추출 단계가 필요합니다. 원한다면 ECMAScript 5 구문을 사용하여 좀 더 자세한 코드를 직접 작성하고 동일한 의미를 얻을 수 있습니다.

앵귤러2는 데코레이터(결국 자바의 애노테이션과 같음)를 사용하여 선언적으로 DI를 설정한다.

데코레이터는 ES7에서 제안하므로 현재 브라우저가 지원하는 ES5로에 트랜스파일링이 필요하다.

## Server-side rendering

서버 측 렌더링의 또 다른 일반적인 사용 사례는 검색 엔진 최적화(Search Engine Optimization)에 친숙한 응용 프로그램을 작성하는 데 있습니다. AngularJS 1.x 응용 프로그램을 검색 엔진으로 색인화 할 수 있게 하기 위해 과거에 사용 된 몇 가지 해킹이 있었습니다. 예를 들어, headless 브라우저로 각 페이지의 스크립트를 실행하고 렌더링 된 결과를 HTML 파일로 캐시하여 검색 엔진이 액세스 할 수 있도록하는 응용 프로그램을 가로챌 수 있습니다. Angular 2를 DOM과 분리하면 Angular 2 애플리케이션을 브라우저 컨텍스트 외부에서 실행할 수 있습니다.

화면구성을 클라이언트에서 행하게 되면 서치엔진에 컨텐츠가 노출되지 않는다. 앵귤러2를 서버사이드에서 처리해야만 서치엔진에 컨텐츠가 노출된다. 앵귤러2를 서포트하는 유니버설 앵귤러를 사용하여 이를 처리할 수 있다.

## TypeScript

Angular 코어 팀은 가능한 더 나은 툴링과 컴파일 타임 형식 검사를 통해 TypeScript를 사용하기로 결정했습니다. 이 기능은 생산성을 높이고 오류를 발생시키지 않는 데 도움이됩니다. 앞의 그림에서 보듯이 TypeScript는 ECMAScript의 상위 집합입니다. 그것은 명시 적 타입 주석과 컴파일러를 소개합니다. TypeScript 언어는 오늘날의 브라우저에서 지원되는 일반 JavaScript로 컴파일됩니다. 1.6 버전부터 TypeScript는 ECMAScript 2016 데코레이터를 구현하므로 Angular 2에 완벽한 선택입니다.

타입스크립트는 ECMAScript의 슈퍼셋(상위집합)이다. 타입스크립트를 사용하면 컴파일타임에 타입체크가 가능해서 현저히 에러를 줄일 수 있다. 또한 IDE들을 사용하는 경우 타입정보를 바탕으로 개발자의 코드작성을 도울 수 있다.

## Templates

템플릿은 AngularJS 1의 핵심 기능 중 하나입니다. 그들은 단순한 HTML이며 mustache 와 같은 대부분의 템플릿 엔진과는 달리 중간 처리 및 컴파일을 필요로하지 않습니다. AngularJS의 템플리트는 사용자 정의 요소 및 속성을 사용하여 내부 DSL (Domain Specific Language)을 작성하여 HTML을 확장 할 수 있으므로 단순함의 힘을 더합니다.

예를 들어, 지시문을 작성한 후 사용자가 속성을 사용하여 속성을 전달할 수 있도록하려는 경우를 가정 해 보겠습니다. AngularJS 1.x에서는 다음과 같은 세 가지 방법으로 접근 할 수 있습니다.

```
< user name =" literal" > </ user >
< user name =" expression" > </ user >
< user name ="{{ interpolate}}" > </ user >
```

## ng-for

For instance, if we want to iterate over a list of users and display their names in AngularJS 1.x, we can use:

```
< div ng-for =" user in users" >{{ user.name}} </ div >
```

Although this syntax looks intuitive to us, it allows limited tooling support.

However, Angular 2 approached this differently by bringing a little bit more explicit syntax with richer semantics:

```
< template ngFor let-user [ngForOf] =" users" > {{ user.name}}</ template >
```

However, this syntax is too verbose for typing. Developers can use the following syntax, which later gets translated to the more verbose one:

```
< li *ngFor =" let user of users" > {{ user.name}} </ li >
```

## Detection Mechanisms

원본: <http://dontpanic.42.nl/>

## Bindings

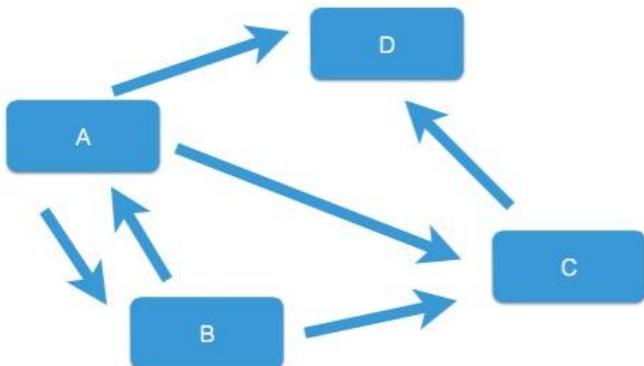
Angular는 뷰에서 자동으로 값을 업데이트합니다. Angular 2.0 팀은 시스템을보다 빠르게 만드는 데 많은 노력을 기울였습니다. 팀은 Angular 1.x 보다 3배에서 10배 빠른 속도 증가를 보고합니다. 그러나 이 속도 증가가 가능하려면 근본적으로 변화해야 합니다.

## Bindings in Angular 1.x

Angular 2.0 이 1.x 보다 빠른 이유와 Angular 2.0 팀이 어떻게 했는지 이해하기 전에 먼저 Angular 1.x 가 바인딩을 처리하는 방법을 살펴보아야 합니다.

## An Angular 1.x App

A, B, C, D 의 네 가지 바인딩을 가진 가상의 Angular 1.x 앱에서 시작해 보겠습니다. 이러한 바인딩은 아래 이미지에서 묘사 된 것처럼 서로 관계가 있습니다.



위의 이미지는 바인딩 A가 바인딩 B와 관계가 있고 반대의 경우도 있음을 보여줍니다. 이것은 A가 무언가를 업데이트 할 때마다 구성 요소 B가 변경될 수 있지만 꼭 그렇게 할 필요가 없을 수도 있음을 의미합니다. 반대의 경우도 마찬가지입니다. B를 변경하면 A가 업데이트되어야 할 수도 그렇지 않을 수도 있습니다.

일방적인 관계가 있을 수도 있습니다. 예를 들어, A는 D에만 영향을 미치지만 반대 방향으로는 영향을 미치지 않습니다.

바인딩은 간접적으로 다른 바인딩과 미묘한 관계를 가질 수도 있습니다. B는 바인딩 C를 통해 바인딩 D와 관계가 있

습니다.

요점은 Angular 1.x의 바인딩 간의 관계가 꽤 복잡해 질 수 있다는 것입니다. 그렇다면 Angular 1.x는 바인딩 및 UI 갱신을 어떻게 파악할까요? 대답은 dirty checking입니다.

## Dirty Checking

더티체킹은 다음과 같이 설명 할 수 있습니다. 보기가 변경 될 때마다, 사용자가 무언가를 클릭 할 때마다, \$http 요청이 완료 될 때마다, Angular가 각 바인딩의 값을 확인하고 이전 값과 비교합니다.

Angular가 값을 변경하면 UI가 업데이트 됩니다. 값이 다를 때 '더티'로 간주되므로 '더티체킹'이라는 용어가 사용됩니다. Angular가 변경 사항을 확인하는 단계를 "다이제스트 단계"라고 합니다.

이 예제를 좀 더 구체적으로 만들어 보면 :

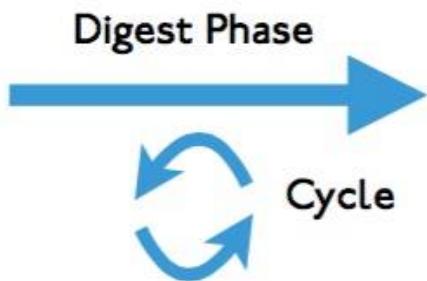
"age"라는 변수가 있고 그 값은 현재 16이라고 가정 해 봅니다. 그러면 어떤 이벤트가 다이제스트 단계를 트리거합니다. 이제 "age"의 새 값은 17입니다. Angular는 16과 17을 비교하고 변경이 발생했으며 UI를 업데이트한다고 합니다.

하지만 다른 바인딩에 종속 된 바인딩을 가지고 있다면 어떻게 업데이트될까요? 우리는 인간으로서 두 바인딩 사이의 '관계'가 무엇인지, 다른 것은 어느 것에 의존 하는지를 알고 있으므로 바인딩을 평가할 순서를 직관적으로 알 수 있습니다.

그러나 Angular는 어떤 바인딩을 먼저 평가해야하는지 어떻게 알 수 있습니까? 대답은 Angular 1.x가 '바인딩'간의 관계에 대해 아무것도 모르는 것입니다. 그래서 더티체킹 작업을 수행 할 순서를 알 수 없습니다. Angular가 대신 모든 바인딩이 안정화 될 때까지 각 바인딩을 평가합니다.

Angular는 바인딩이 안정화 될 때까지 각 바인딩을 평가하는 "루프"를 실행하여 이 작업을 수행합니다.

이 "루프"는 다이제스트 단계의 일부인 다이제스트 사이클이라고 합니다. 다이제스트 사이클은 모든 바인딩이 순환 실행 사이에 변경 사항을 보고하지 않을 때까지 모든 바인딩을 해결합니다. 다이제스트 사이클은 다이제스트 단계의 하위 부분입니다.



관계가 있는 두 개의 바인딩이 있는 경우 두 바인딩이 더 이상 변경되지 않기 전에ダイジェスト주기를 여러 번 실행해야 할 수도 있습니다. 10 사이클 후에 바인딩들이 안정되지 않으면 Angular가 포기하고 오류가 발생합니다. 이 마법적인 리미트인 10을 생존시간(Time To Live)이라고 하며 원하는 경우 늘리거나 줄일 수도 있습니다.

ダイジェスト주기가 완료되고 시스템이 안정적인 디제스트 단계로 보고되면 Angular는 보기를 다시 렌더링합니다.

결론적으로 말하자면, Angular 1.x가 바인딩 간의 실제 '의미'가 의미 상 의미하는 바를 알지 못하지만 여전히 UI의 해당 관계를 올바르게 업데이트하는 것은 디제스트주기입니다.

## What's wrong with 1.x bindings?

바인딩을 해결하는 Angular 1.x 방식과 그 복잡한 관계는 매우 복잡한 문제를 해결하는 정말 좋은 방법입니다. 그러나 1.x 방식에는 세 가지 단점이 있습니다.

### Expensive

디제스트주기에서 바인딩을 확인하여 복잡한 관계가 있는 바인딩을 해결하면 차선책의 성능이 발생할 수 있습니다.

바인딩간에 여러 복잡한 관계가 있는 매우 복잡한 응용 프로그램을 사용하는 경우 시스템이 안정화되었음을 디제스트 단계에 보고하기 전에 Angular 1.x의 디제스트 사이클 다중 루프가 필요할 수 있습니다.

이러한 의미에서 바인딩 바인딩은 잠재적으로 매우 비쌉니다.

### Unpredictable

당신이 나에게 다음과 같은 템플릿을 주었다면 시스템은 또한 예측할 수 없다.

```
<div ng-controller="PersonController as personController">...</div>
  <h1>Hi {{ personController.person.name }}</h1>
  <person-view person="personController.person"></person-view>
</div>
```

그리고 당신은 저에게 "이 템플릿에서 바인딩이 어떻게 해결 되나요?"라고 묻습니다. 나는 곧바로 답을 줄 수 없을 것입니다. 응답을 제공하기 전에 "PersonController"와 "personView" 지시문에 깊이 들어가야합니다.

예를 들어, 어떤 유형의 `bing`이 `personView`의 "person"입니까? 내 대답은 바인딩의 유형에 따라 다르 겠지만 Angular가 실제로 디제스트 단계에서 바인딩을 해결하는 방법을 말할 수도 없습니다.

기본적으로 Angular 1.x 바인딩 시스템은 결정적이지 않습니다. 동일한 입력을 주면 매번 결과에 도달하는 데 다른 경로가 필요할 수 있습니다. 이 시스템의 특성상 Angular 1.x 응용 프로그램을 추론하기가 어렵습니다.

## **Unnecessary**

### **Immutable**

바인딩이 불필요한 경우도 있습니다. 다음을 고려하십시오. 데이터 구조가 변경되지 않는다는 것을 알고 있다면 어떻게 될까요?

예를 들어, 문자열 배열을 기반으로 메뉴를 렌더링 한 경우 배열이 변경되지 않을 가능성이 있다는 것을 알고 있었습니다. 즉, 응용 프로그램을 실행하는 동안 절대로 메뉴를 변경하지 않는다는 것을 알고 있습니다.

그러한 불변(변경하지 않는) 데이터 구조에 대한 더러운 검사는 순수한 시간 낭비입니다. Angular 1.x에서는 이 구조가 디자인 단계에서 제외되어야한다고 말할 수 있는 방법이 없기 때문에 매번 평가됩니다.

### **Observable**

또 다른 상황은 특정 이벤트가 발생할 때만 변경되는 구성 요소가 있을 수 있다는 것입니다. 즉, 해당 이벤트가 발생하지 않으면 오브젝트가 변경되지 않습니다.

다시 Angular 1.x에는 그러한 구성 요소가 있음을 시스템에 알리는 방법이 없습니다. 시스템은 인간이 그것이 쓸데없다는 것을 알더라도 그 구성 요소를 더럽힐 것입니다.

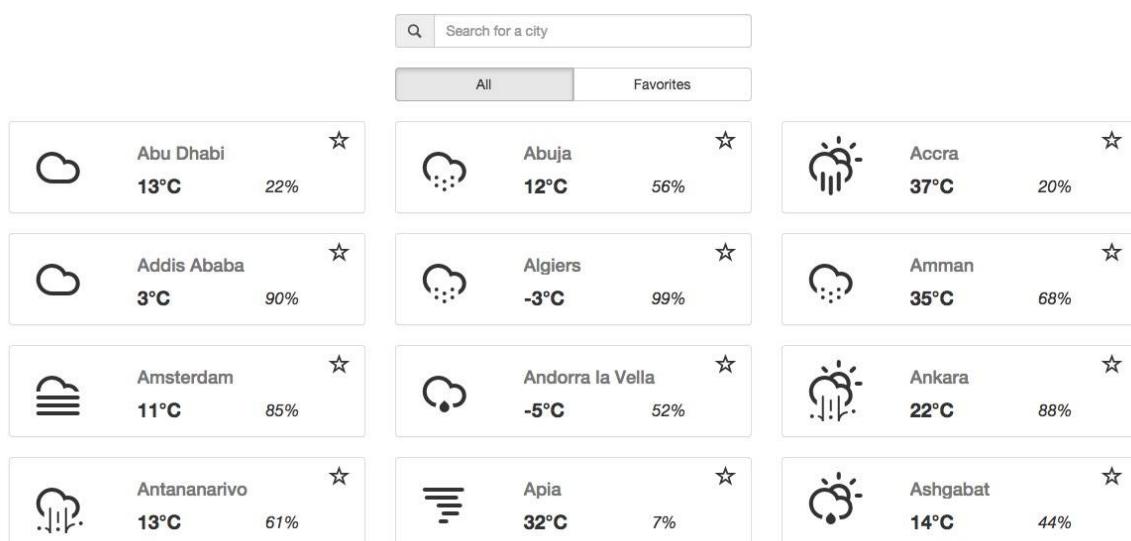
## Bindings in Angular 2.0

이제 Angular 1.x에서 바인딩을 처리하는 방법을 알았고 결함의 일부를 알고 있으므로 Angular 2.0이 이러한 결함을 완화하고 시스템을 개선하는 방법을 살펴볼 수 있습니다. 앞서서 우리는 Angular 2.0이 컴포넌트 기반 프레임 워크가 된다는 것을 알았기 때문에 컴포넌트 기반이 바인딩 시스템에 어떻게 영향을 미치는지 알기 위해서 먼저 Angular 2.0 응용 프로그램의 해부도를 조사해야합니다.

## Anatomy of an Angular 2.0 application

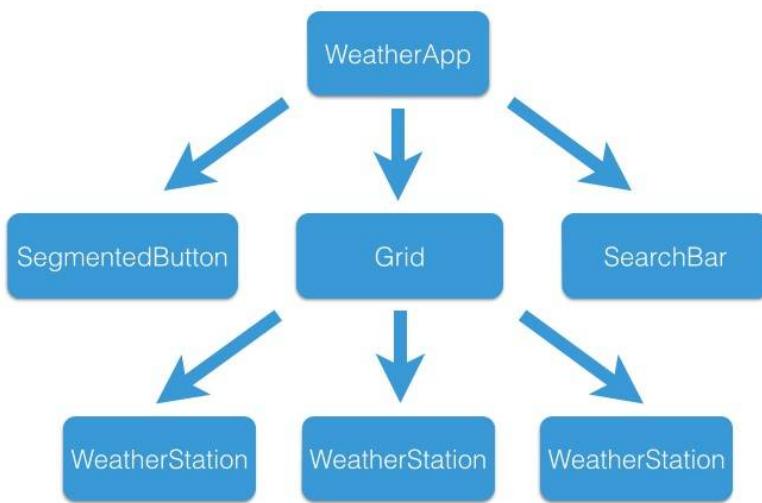
우리는 세계의 여러 도시의 날씨 정보를 제공하는 응용 프로그램이 있다고 가정 해 보겠습니다. 응용 프로그램은 다음과 같이 보입니다.

### Weather



응용 프로그램은 WeatherStation의 그리드로 구성되어 있습니다. 각 스테이션은 이름, 온도, 습도 및 날씨의 현재 상태를 나타내는 아이콘으로 구성됩니다. "star"아이콘을 클릭하면 기상 관측소를 즐겨 볼 수 있습니다. 그리드 위에는 사용자가 이름을 기반으로 방송국을 필터링 할 수 있는 SearchBar와 모든 방송국과 사용자가 좋아하는 방송국간에 전환 할 수있는 SegmentedButton이라는 두 개의 막대가 있습니다.

이 응용 프로그램은 Angular 2.0으로 작성되었으므로 컴포넌트를 기반으로합니다. 컴포넌트는 조립이 가능합니다. 이는 컴포넌트가 서로의 내부에 중첩 될 수 있음을 의미합니다. 날씨 앱의 구조는 다음과 같습니다.



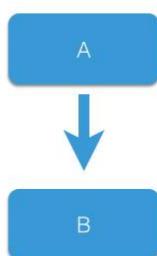
각 구성 요소는 하나의 "루트"구성 요소 (이 경우 WeatherApp 구성 요소)의 직접 또는 간접 자손입니다. 이것은 중요한 실현을 가져옵니다. Angular 2.0 어플리케이션은 나무입니다!

## What is so great about trees?

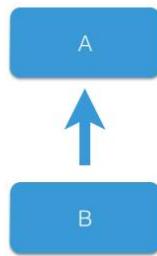
구성 요소 간의 관계가 즉시 명백하기 때문에 나무를 매우 쉽게 이해할 수 있습니다. Angular 1.x의 바인딩 사이의 관계 이미지를 날씨 어플리케이션 트리의 이미지와 비교하면 Angular 1.x의 관계는 신속하게 제어 할 수 없게 됩니다.

Angular 2.0에는 두 가지 컴포넌트 사이에 한 가지 유형의 관계만 존재합니다. 컴포넌트는 다른 컴포넌트의 부모이거나 해당 컴포넌트의 하위 요소 일 수 있습니다.

그 관계에서 부모 컴포넌트는 하위 컴포넌트로 정보를 보낼 수 있으며 하위 컴포넌트는 부모에게 이벤트를 다시 보낼 수 있습니다. Angular 2.0의 컴포넌트 간의 관계는 다음과 같이 성문화 될 수 있습니다.



`bindings []  
go down`



`events ()  
go up`

이 속성을 사용하면 템플릿에서 요소를 발견 할 때 컴포넌트를 매우 쉽게 추론 할 수 있습니다.

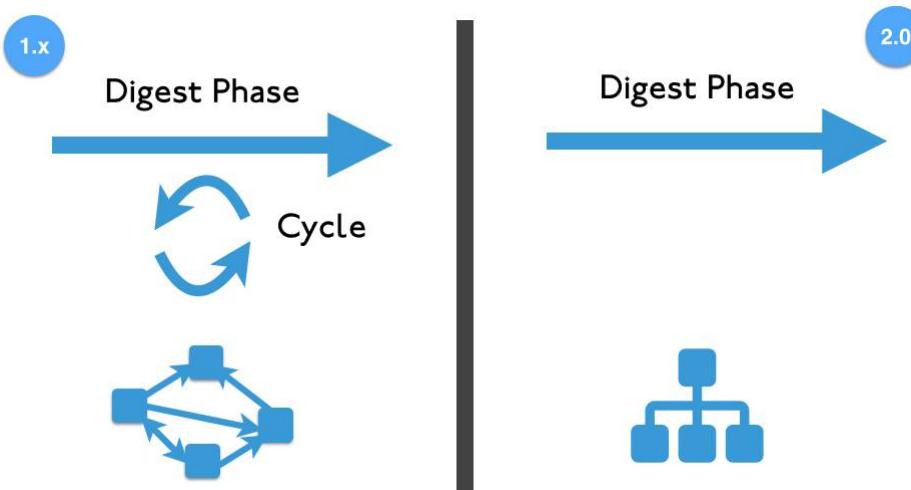
```
<grid>
  <div *for="#station of stations">
    <station [station]="station" (station-changed)="stationsDidChange()"></station>
  </div>
</grid>
```

템플릿에서 [station] 바인딩이 상위 (이 경우 그리드의 "스테이션"속성)에서 나옵니다. Grid가 Station 컴포넌트의 부모이기 때문에 Grid 컴포넌트에서 "(station-changed)"이벤트가 "stationDidChange ()"를 호출한다는 것도 간단하게 추론할 수 있습니다. Angular 2.0에서는 템플릿을 읽고 컴포넌트 간의 관계를 즉시 이해할 수 있습니다.

Angular 2.0 응용 프로그램이 트리라는 사실은 다이제스트 단계가 작동하는 방식에도 영향을 미칩니다. 2.0에서는 다이제스트주기가 더 이상 필요하지 않습니다. 왜냐하면 Angular가 모든 바인딩을 해결하기 위해서는 트리의 맨 위에서 트리의 맨 아래로 한 번만 처리하면 되기 때문입니다.

그 이유는 간단합니다. 컴포넌트는 상위 컴포넌트에서만 데이터([] 바인딩)를 받을 수 있습니다. 하위 컴포넌트는 상위 바인딩이 해결 될 때 해당 바인딩을 평가할 수 있습니다. 따라서 아이는 부모를 기다려야합니다. 바인딩은 아래로 내려 가기 때문에 부모는 자식으로부터 데이터를 받을 수 없습니다. 이는 자식 컴포넌트 바인딩이 부모에 영향을 줄 수 없다는 것을 의미합니다. 다시 말해 Angular는 트리의 "바닥"에 도달하면 됩니다.

여기서 Angular 1.x와 Angular 2.0의 변경 감지를 시각적으로 대조 할 수 있습니다.



Angular 1.x의 바인딩 시스템은 가격이 비싸고 예측할 수 없으며 불필요한 것으로 나타났습니다.

이제 Angular 2.0에 대해 알고 있는 점을 감안하면 시스템이 비싸지 않다는 것을 말할 수 있습니다.

더 이상 데이터주기가 없기 때문에 최적으로 작동 할 수 있습니다. 이 시스템은 데이터와 이벤트의 흐름이 명확하게 정의되어 있기 때문에 현재 예측 가능합니다. 따라서 우리는 인간이 그것을 더 잘 추론 할 수 있습니다.

그러나 Angular 2.0이 "불필요한"것에 대해 우리가 필요로 하지 않는 일을하는 것을 막을 수 있습니까?

네 가능합니다!

## Change Detection

앵귤러 2.0에서는 Angular가 컴포넌트별로 탐지를 변경하는 방식을 대신 사용할 수 있습니다. 이를 통해 Angular 2.0이 절대적으로 필요할 때 더 많은 성능을 낼 수 있습니다.

기본적으로 Angular 2.0은 런타임에 각 컴포넌트에 대한 변경 감지기 "클래스"를 생성합니다. WeatherStation Angular 2.0이라는 컴포넌트가 있으면 WeatherStation\_ChangeDetector 클래스가 생성됩니다.

이 클래스는 컴포넌트에 대한 "메타"데이터를 읽고 모든 입력 및 출력을 읽고 더티 검사를 수행 할 클래스를 생성합니다. 따라서 컴포넌트에 대한 모든 입출력을 설명해야합니다.

예를 들어 WeatherStation에 "온도"속성 만있는 경우이 클래스는 다음과 같이 보일 수 있습니다.

```
var temperature = obj.temperature;
if (temperature !== this.previousTemperature) {
    this.previousTemperature = temperature;
    this.weatherStation.temperature = temperature;
}
```

Angular 2.0이 모든 컴포넌트에 대해 이러한 "클래스"를 생성하는 이유는 JavaScript 가상 시스템(VM)이 "일반"코드를 최적화 할 수 있는 것보다 특정 "코드"방법을 최적화 할 수 있기 때문입니다.

매우 기술적 인 측면에서 VM은 다형성 코드보다 단일 형 코드를 더 잘 최적화 할 수 있습니다.

Vyacheslav Egorov의 위대한 블로그 포스트는 이것이 사실 인 이유를 설명합니다.

<http://mrale.ph/blog/2015/01/11/whats-up-with-monomorphism.html>

Angular 1.x는 다형성 검사 알고리즘을 사용했기 때문에 VM에서 잘 최적화 할 수 없었습니다.

Angular에게 \_ChangeDetector 클래스를 직접 구현하려고한다고 말할 수 있습니다. 이를 통해 불변의 컴포넌트와 관찰 가능한 컴포넌트를 쓸 수 있습니다. 실제로 "변경 불가능한"동작은 Angular 2.0과 함께 제공됩니다.

불변의 컴포넌트를 만들려면 다음과 같이하면됩니다 :

```
@Component({
```

```
changeDetector: ON_PUSH  
})
```

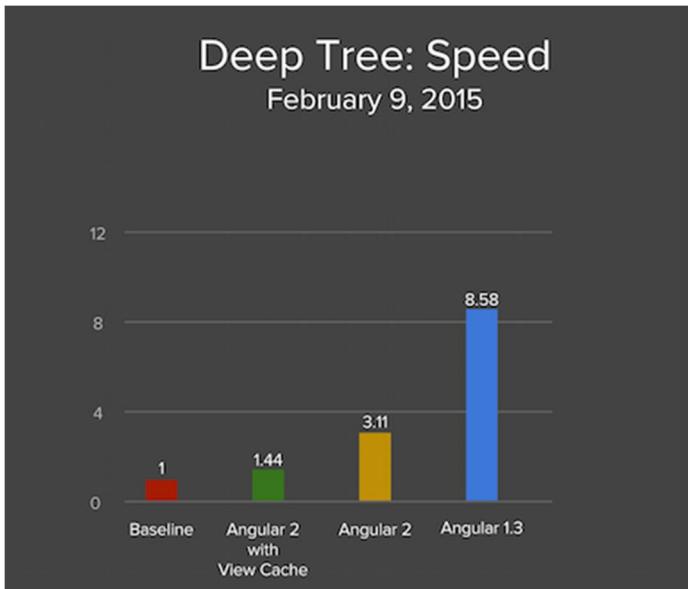
즉, 새 바인딩이 구성 요소로 푸시 될 때 @Component는 변경 감지만 실행합니다.

따라서 구성 요소가 새로운 바인딩을 수신하지 못하면 절대로 디제스트 단계에 포함되지 않습니다. Angular 2.0은 변경되지 않을 것으로 판단되는 시간을 낭비하지 않습니다.

구성 요소마다 변경 감지 작업 방식을 설정할 수 있다는 멋진 점은 다양한 전략을 혼합하고 일치시킬 수 있다는 것입니다. 응용 프로그램의 일부는 불변일 수 있고, 일부는 "기본값"일 수 있으며 일부는 이국적인 전략을 제시 할 수 있습니다. 이렇게하면 Angular가 "불필요한" 일을 하지 못하게하는 강력한 도구를 갖게 됩니다.

이 사고에서 벗어난다면 성능을 높이기 위해 자신의 "변화 감지" 알고리즘을 선언해야합니다. Angular 2.0 응용 프로그램이 나무라는 사실만으로도 성능 향상에 도움이됩니다. 당신은 자신의 전략을 정의하는 것을 분명히 할 수 있으며 여전히 빠를 것입니다.

## Speed



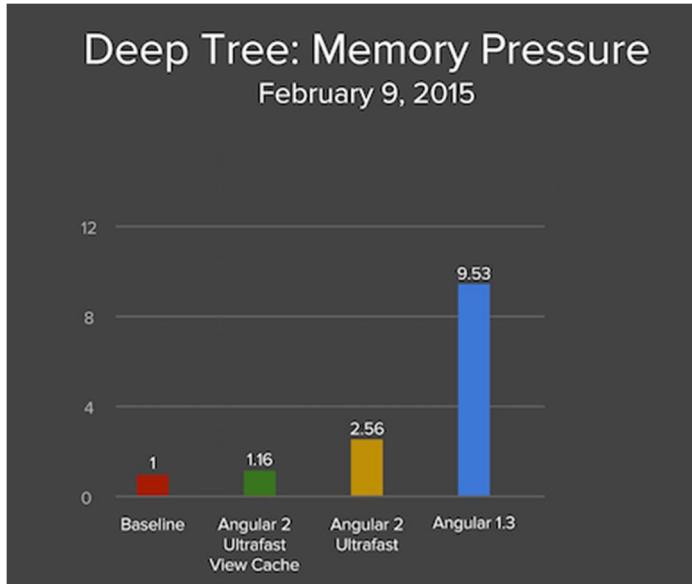
그림은 다양한 방법으로 작성된 동일한 응용 프로그램의 성능을 보여줍니다. 왼쪽에는 바닐라 자바 스크립트로 작성된 기본 애플리케이션인 빨간색 막대가 있습니다. 이 기본 애플리케이션은 상상할 수 있는 가장 최적화된 (그러나 추악한) JavaScript로 작성되었습니다. 그것은 제로 수준의 추상화를 가지고 있으며, 추상화의 모든 수준은 보다 적은 속도의 형태로 가격이 제공되므로 HTML 응용 프로그램을 작성하는 가장 빠른 방법입니다. Angular 2.0 팀은 이 기본 애플리케이션을 사용하여 얼마나 빨리 얻을 수 있는지 확인합니다.

오른쪽의 파란색 막대는 앵귤러 1.3을 나타내며 기준선보다 8.58 배 느립니다. 그 옆에 주황색의 Angular 2.0이 있는

데 이는 "신선한" Angular 2.0 응용 프로그램을 나타냅니다. 그 옆에 "뜨거운" 상태의 Angular 2.0을 나타내는 "녹색" 막대가 있습니다. 이는 일부 뷰를 캐시했음을 의미합니다.

"신선한" Angular 2.0은 1.x보다 3 배 빠릅니다. 더 좋은 점은 Angular 2.0 응용 프로그램을 클릭 할수록 속도가 빠르며 2배 이상 빠릅니다. 앵귤러 2.0은 자동으로 뷰 캐싱을 제공합니다.

## Memory Pressure



메모리 효율성은 우리가 살고있는이 모바일 세계에서 점점 더 중요 해지고 있습니다. 모바일 장치는 데스크톱 사춘 만큼 많은 메모리를 가지고 있지 않습니다. Angular 2.0은 모바일의 첫 번째 프레임 워크라는 점에서 자부심을 가지고 있으므로 메모리 소비를 심각하게 고려해야합니다.

## Directive

Angular 2의 지시문의 주요 목적은 ES2015 (ES6) 클래스에 정의 된 사용자 정의 논리로 DOM을 확장하여 동작을 DOM에 첨부하는 것입니다. 우리는 이 클래스들을 지시어와 관련된 컨트롤러로 생각할 수 있으며, 생성자를 AngularJS 1의 지시어 링크 기능과 유사하다고 생각할 수 있습니다. 그러나 새로운 지시문에는 구성 가능성이 제한되어 있습니다. 템플릿의 정의를 허용하지 않으므로 지시문을 정의하는 데 이미 알려진 특성 대부분이 필요하지 않습니다.

## Components VS Directive

### Components

- 구성 요소를 등록하기 위해 @Component 메타 데이터 주석을 사용합니다.
- Component 는 쉐도우 DOM 을 사용하여 컴포넌트라고하는 캡슐화 된 시각적 동작을 생성하는 지시어입니다. 컴포넌트는 일반적으로 UI 위젯을 만드는 데 사용됩니다.
- 컴포넌트는 응용 프로그램을 더 작은 컴포넌트로 분해하는 데 사용됩니다.
- DOM 요소 당 하나의 컴포넌트만 존재할 수 있습니다.
- @View decorator 또는 templateUrl 템플릿은 구성 요소에서 필수입니다.

### Directive

- 디렉티브를 등록하기 위해 @Directive 메타 데이터 주석을 사용합니다.
- 디렉티브는 기존 DOM 요소에 **behavior** 를 추가하는 데 사용됩니다.
- 디렉티브는 재사용 가능한 부품을 설계하는 데 사용됩니다.
- DOM 엘리먼트 별로 많은 디렉티브를 사용할 수 있습니다.
- Directive 는 View 를 사용하지 않습니다.

## Step 2 – 분석 : Registration and Login

잘 짜여진 코드 분석을 통해 새로운 기술을 쉽게 받아들일 수 있는 연습을 해 보자.



<http://jasonwatmore.com/post/2016/09/29/angular-2-user-registration-and-login-example-tutorial>

```
git clone https://github.com/cornflourblue/angular2-registration-login-example.git
cd angular2-registration-login-example
npm install
```

## Step 3 - 실습 : Tour of Heroes Tutorial

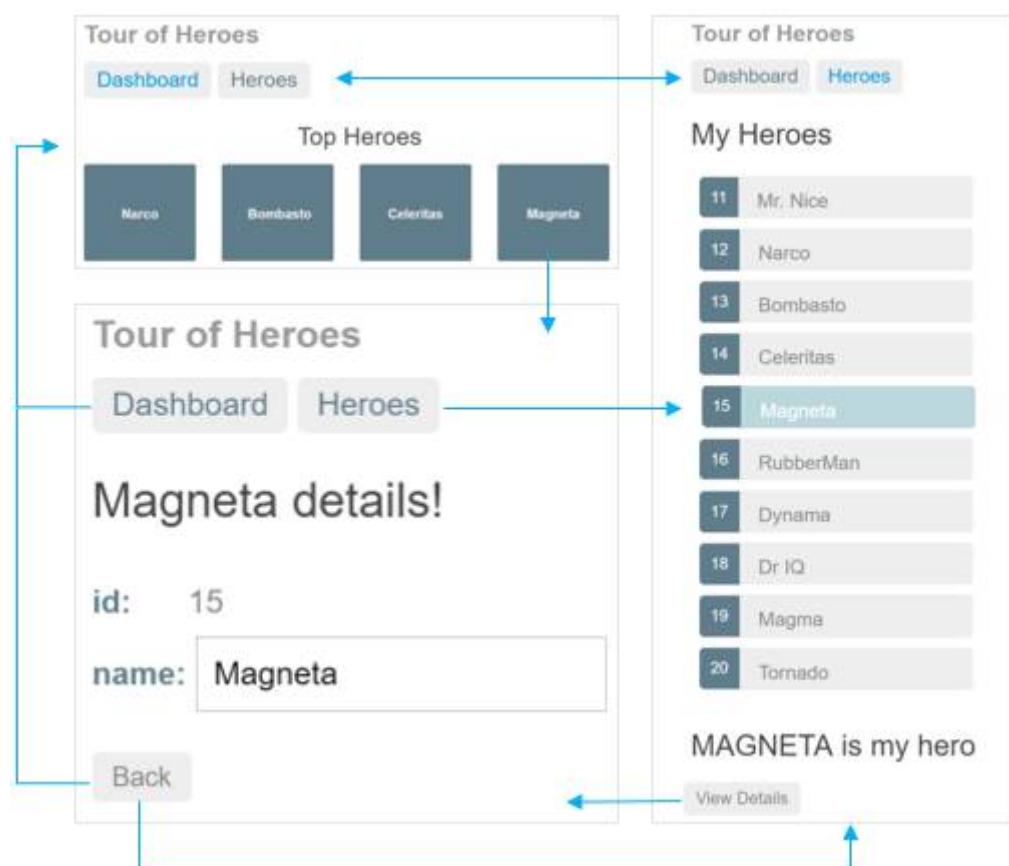
참고 : <https://angular.io/tutorial>

이 튜토리얼의 가장 큰 계획은 인력 관리 기관이 영웅의 안정을 관리하는 데 도움이되는 앱을 개발하는 것입니다.

Tour of Heroes 앱은 Angular의 핵심 기본 사항을 다룹니다. 영웅 목록을 얻고 표시하고, 영웅의 세부 사항을 편집하고, 다양한 관점에서 탐색하는 등, 본격적인 데이터 기반 앱에서 기대할 수 있는 많은 기능을 갖춘 기본 앱을 구축하게됩니다.

빌트인 디렉티브를 사용하여 요소를 표시하거나 숨기고 영웅 데이터 목록을 표시합니다. 영웅의 세부 정보를 표시하고 영웅의 배열을 표시하는 컴포넌트를 만듭니다. 읽기 전용 데이터에 단방향 데이터 바인딩을 사용합니다. 편집 가능한 필드를 추가하여 양방향 데이터 바인딩으로 모델을 업데이트합니다. 키 스트로크 및 클릭과 같은 사용자 이벤트에 구성 요소 메소드를 바인딩합니다. 사용자는 마스터 목록에서 영웅을 선택하고 세부보기에서 해당 영웅을 편집 할 수 있습니다. 파이프로 데이터 서식을 지정합니다. 영웅들을 모으기 위해 공유 서비스를 만들 것입니다. 또한 라우팅을 사용하여 다양한보기와 해당 구성 요소를 탐색합니다.

Angular가 시작하기에 충분한 코어 앵귤레이션을 배우고 자신이 필요한 모든 것을 할 수 있다는 자신감을 얻습니다.



## 01 : The Hero Editor

- 영웅 투어는 앱의 제목과 영웅 개체의 속성을 표시하는 이중 중괄호 보간법 (단방향 데이터 바인딩 유형)을 사용합니다.
- ES2015의 문자열 리터럴을 사용하여 여러 줄 템플릿을 작성하여 템플릿을 읽을 수 있게 만듭니다.
- 내장 ngModel 지시문을 사용하여 <input> 요소에 양방향 데이터 바인딩을 추가합니다. 이 바인딩은 영웅의 이름을 표시하고 사용자가 변경할 수 있도록 합니다.
- ngModel 지정 문은 hero.name 의 다른 모든 바인딩에 변경 사항을 전파합니다.

```
ng new angular-tour-of-heroes --routing=true
```

### package.json

```
{  
  "name": "angular-tour-of-heroes",  
  "version": "0.0.0",  
  "license": "MIT",  
  "scripts": {  
    "ng": "ng",  
    "start": "ng serve --open",  
    "build": "ng build",  
    "test": "ng test",  
    "lint": "ng lint",  
    "e2e": "ng e2e"  
},  
  "private": true,  
  "dependencies": {  
    "@angular/animations": "^4.0.0",  
    "@angular/common": "^4.0.0",  

```

```

"@angular/cli": "1.2.4",
"@angular/compiler-cli": "^4.0.0",
"@angular/language-service": "^4.0.0",
"@types/jasmine": "~2.5.53",
"@types/jasminewd2": "~2.0.2",
"@types/node": "~6.0.60",
"codemlyzer": "~3.0.1",
"jasmine-core": "~2.6.2",
"jasmine-spec-reporter": "~4.1.0",
"karma": "~1.7.0",
"karma-chrome-launcher": "~2.1.1",
"karma-cli": "~1.0.1",
"karma-coverage-istanbul-reporter": "^1.2.1",
"karma-jasmine": "~1.1.0",
"karma-jasmine-html-reporter": "^0.2.2",
"protractor": "~5.1.2",
"ts-node": "~3.0.4",
"tslint": "~5.3.2",
"typescript": "~2.3.3"
}
}

```

다음 명령으로 개발서버를 기동합니다.

```

cd angular-tour-of-heroes
npm start

```

```

> angular-tour-of-heroes@0.0.0 start C:\Lesson\codebase\angular\source\chapter4\angular-tour-of-heroes
> ng serve --open

** NG Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200 **
11% building modules 9/13 modules 4 active ...gular\platform-browser-dynamic.es5.jswebpHash: 85de103fed07f8ee68f9
Time: 9877ms

chunk {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 178 kB {4} [initial] [rendered]
chunk {1} main.bundle.js, main.bundle.js.map (main) 4.81 kB {3} [initial] [rendered]
chunk {2} styles.bundle.js, styles.bundle.js.map (styles) 10.5 kB {4} [initial] [rendered]
chunk {3} vendor.bundle.js, vendor.bundle.js.map (vendor) 2.48 MB [initial] [rendered]
chunk {4} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry] [rendered]

webpack: Compiled successfully.

```

### app.component.ts

```

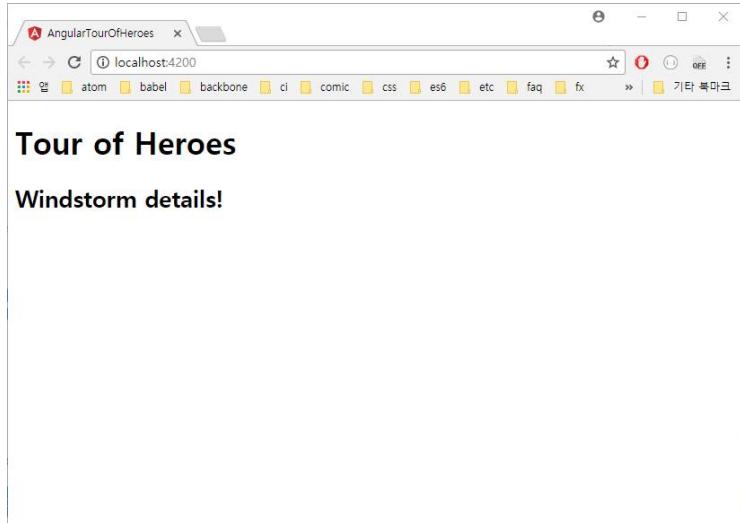
import { Component } from '@angular/core';

```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Tour of Heroes';
  hero = 'Windstorm';
}
```

### app.component.html

```
<h1>{{title}}</h1>
<h2>{{hero}} details!</h2>
```



### app.component.ts

```
import { Component } from '@angular/core';

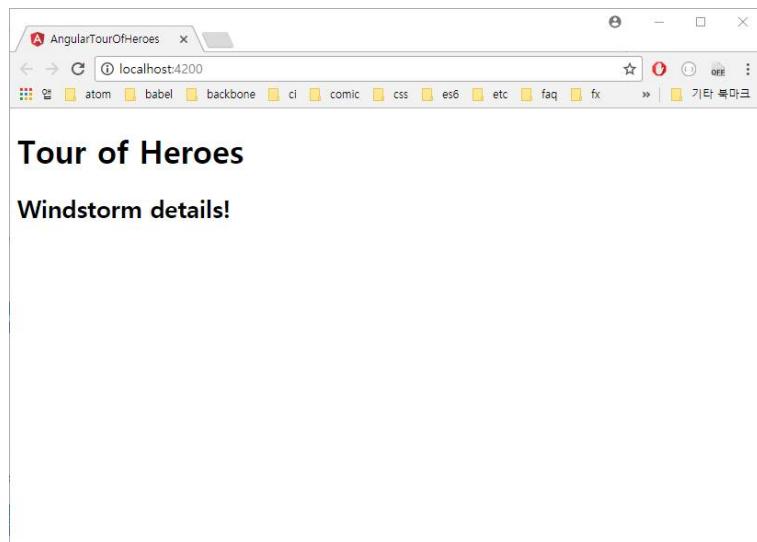
export class Hero {
  id: number;
  name: string;
}

@Component({
  selector: 'app-root',
```

```
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Tour of Heroes';
  // hero = 'Windstorm';
  hero: Hero = {
    id: 1,
    name: 'Windstorm'
  };
}
```

### app.component.html

```
<h1>{{title}}</h1>
<!-- <h2>{{hero}} details!</h2> -->
<h2>{{hero.name}} details!</h2>
```



### app.component.html

```
<h1>{{title}}</h1>
<h2>{{hero.name}} details!</h2>
<div><label>id: </label>{{hero.id}}</div>
<div><label>name: </label>{{hero.name}}</div>
```

## app.component.html

```
<h1>{{title}}</h1>
<h2>{{hero.name}} details!</h2>
<div><label>id: </label>{{hero.id}}</div>
<!-- <div><label>name: </label>{{hero.name}}</div> -->
<div>
  <label>name: </label>
  <input [(ngModel)]="hero.name" placeholder="name">
</div>
```

불행하게도 이 변경 직후에 애플리케이션이 중단됩니다. 브라우저 콘솔에서 보면 Angular가 "ngModel ... isn't a known property of input."라는 불만을 보게됩니다.

NgModel은 유효한 Angular 지시문이지만 기본적으로 사용할 수 없습니다. 선택적으로 사용하는 FormsModule에 속합니다. 해당 모듈을 사용하도록 설정해야 합니다.

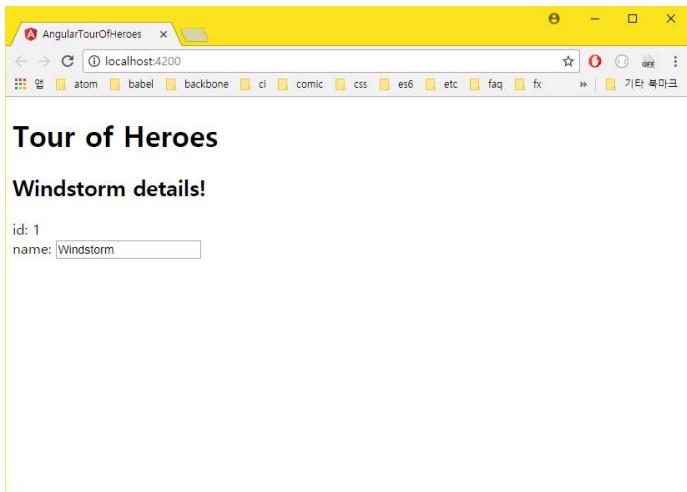
## app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';

import { FormsModule }   from '@angular/forms'; // <-- NgModel lives here

import { AppComponent } from './app.component';

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule // <-- import the FormsModule before binding with [(ngModel)]
  ],
  declarations: [
    AppComponent
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



name 인풋박스의 내용을 변경하여 양방향 데이터 바인딩이 이루어지는지 확인하세요.

## 02 : Master/Detail

이제부터 영웅 투어를 확장하여 영웅 목록을 표시하고 사용자가 영웅을 선택하고 영웅의 세부 정보를 표시 할 수 있게 만들어 봅니다.

### app.component.ts

```
import { Component } from '@angular/core';

export class Hero {
  id: number;
  name: string;
}

const HEROES: Hero[] = [
  { id: 11, name: 'Mr. Nice' },
  { id: 12, name: 'Narco' },
  { id: 13, name: 'Bombasto' },
  { id: 14, name: 'Celeritas' },
  { id: 15, name: 'Magneta' },
  { id: 16, name: 'RubberMan' },
  { id: 17, name: 'Dynamite' },
  { id: 18, name: 'Dr IQ' },
  { id: 19, name: 'Magma' },
  { id: 20, name: 'Tornado' }
];

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Tour of Heroes';
  hero: Hero = {
    id: 1,
    name: 'Windstorm'
  };
  heroes = HEROES;
}
```

## app.component.html

```
<h1>{{title}}</h1>
<h2>{{hero.name}} details!</h2>
<div><label>id: </label>{{hero.id}}</div>
<div>
  <label>name: </label>
  <input [(ngModel)]="hero.name" placeholder="name">
</div>
<ul class="heroes">
  <li *ngFor="let hero of heroes">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>
```

## app.component.css

```
.selected {
  background-color: #CFD8DC !important;
  color: white;
}

.heroes {
  margin: 0 0 2em 0;
  list-style-type: none;
  padding: 0;
  width: 15em;
}

.heroes li {
  cursor: pointer;
  position: relative;
  left: 0;
  background-color: #EEE;
  margin: .5em;
  padding: .3em 0;
  height: 1.6em;
  border-radius: 4px;
}

.heroes li.selected:hover {
  background-color: #BBD8DC !important;
  color: white;
```

```

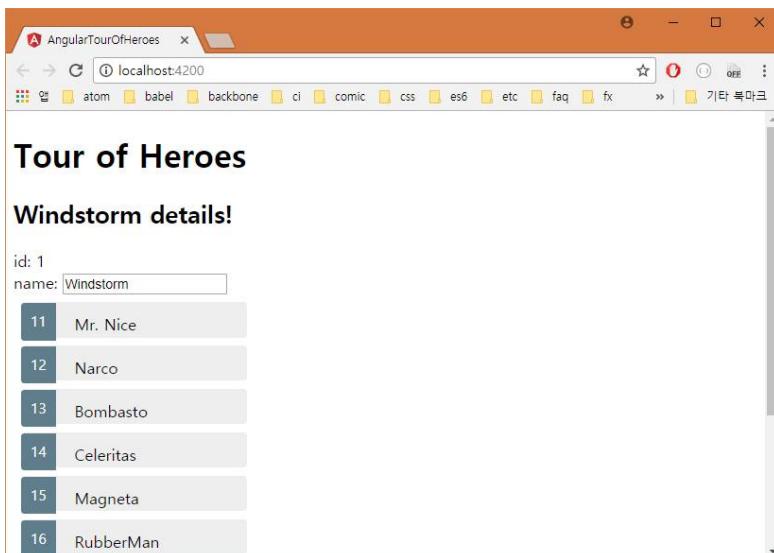
}

.heroes li:hover {
  color: #607D8B;
  background-color: #DDD;
  left: .1em;
}

.heroes .text {
  position: relative;
  top: -3px;
}

.heroes .badge {
  display: inline-block;
  font-size: small;
  color: white;
  padding: 0.8em 0.7em 0 0.7em;
  background-color: #607D8B;
  line-height: 1em;
  position: relative;
  left: -1px;
  top: -4px;
  height: 1.8em;
  margin-right: .8em;
  border-radius: 4px 0 0 4px;
}

```



## app.component.ts

```
import { Component } from '@angular/core';

export class Hero {
  id: number;
  name: string;
}

const HEROES: Hero[] = [
  { id: 11, name: 'Mr. Nice' },
  { id: 12, name: 'Narco' },
  { id: 13, name: 'Bombasto' },
  { id: 14, name: 'Celeritas' },
  { id: 15, name: 'Magneta' },
  { id: 16, name: 'RubberMan' },
  { id: 17, name: 'Dynamite' },
  { id: 18, name: 'Dr IQ' },
  { id: 19, name: 'Magma' },
  { id: 20, name: 'Tornado' }
];

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Tour of Heroes';
  hero: Hero = {
    id: 1,
    name: 'Windstorm'
  };
  heroes = HEROES;

  selectedHero: Hero;
  onSelect(hero: Hero): void {
    this.selectedHero = hero;
  }
}
```

## app.component.html

```
<h1>{{title}}</h1>
<h2>{{hero.name}} details!</h2>
<div><label>id: </label>{{hero.id}}</div>
<div>
  <label>name: </label>
  <input [(ngModel)]="hero.name" placeholder="name">
</div>
<ul class="heroes">
  <li *ngFor="let hero of heroes" (click)="onSelect(hero)">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>
<div *ngIf="selectedHero">
  <h2>{{selectedHero.name}} details!</h2>
  <div><label>id: </label>{{selectedHero.id}}</div>
  <div>
    <label>name: </label>
    <input [(ngModel)]="selectedHero.name" placeholder="name"/>
  </div>
</div>
```

## Tour of Heroes

### Windstorm details!

id: 1  
name:

11	Mr. Nice
12	Narco
13	Bombasto
14	Celeritas
15	Magneta
16	RubberMan
17	Dynama
18	Dr IQ
19	Magma
20	Tornado

### Dynama details!

id: 17  
name:

### app.component.html

```
<h1>{{title}}</h1>
<h2>{{hero.name}} details!</h2>
<div><label>id: </label>{{hero.id}}</div>
<div>
  <label>name: </label>
  <input [(ngModel)]="hero.name" placeholder="name">
</div>
<ul class="heroes">
  <li *ngFor="let hero of heroes" (click)="onSelect(hero)" [class.selected]="hero === selectedHero">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>
<div *ngIf="selectedHero">
  <h2>{{selectedHero.name}} details!</h2>
  <div><label>id: </label>{{selectedHero.id}}</div>
  <div>
    <label>name: </label>
    <input [(ngModel)]="selectedHero.name" placeholder="name"/>
  </div>
```

```
</div>
```

## Tour of Heroes

### Windstorm details!

id: 1  
name:

11	Mr. Nice
12	Narco
13	Bombasto
14	Celeritas
15	Magneta
16	RubberMan
17	Dynama
18	Dr IQ
19	Magma
20	Tornado

### Bombasto details!

id: 13  
name:

### app.component.ts

```
import { Component } from '@angular/core';

export class Hero {
  id: number;
  name: string;
}

const HEROES: Hero[] = [
  { id: 11, name: 'Mr. Nice' },
  { id: 12, name: 'Narco' },
  { id: 13, name: 'Bombasto' },
  { id: 14, name: 'Celeritas' },
  { id: 15, name: 'Magneta' },
  { id: 16, name: 'RubberMan' },
  { id: 17, name: 'Dynama' },
  { id: 18, name: 'Dr IQ' },
```

```

{ id: 19, name: 'Magma' },
{ id: 20, name: 'Tornado' }
];

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Tour of Heroes';
  // hero: Hero = {
  //   id: 1,
  //   name: 'Windstorm'
  // };
  heroes = HEROES;

  selectedHero: Hero;
  onSelect(hero: Hero): void {
    this.selectedHero = hero;
  }
}

```

## app.component.html

```

<h1>{{title}}</h1>
<!-- <h2>{{hero.name}} details!</h2>
<div><label>id: </label>{{hero.id}}</div>
<div>
  <label>name: </label>
  <input [(ngModel)]="hero.name" placeholder="name">
</div> -->
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes" (click)="onSelect(hero)" [class.selected]="hero === selectedHero">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>
<div *ngIf="selectedHero">
  <h2>{{selectedHero.name}} details!</h2>
  <div><label>id: </label>{{selectedHero.id}}</div>
  <div>

```

```
<label>name: </label>
<input [(ngModel)]="selectedHero.name" placeholder="name"/>
</div>
</div>
```

## Tour of Heroes

### My Heroes

11	Mr. Nice
12	Narco
13	Bombasto
14	Celeritas
15	Magneta
16	RubberMan
17	Dynama
18	Dr IQ
19	Magma
20	Tornado

### Magma details!

id: 19

name:

## 03 : Multiple Components

AppComponent는 모든 것을 다 처리하고 있습니다. 처음에는 하나의 영웅에 대한 세부 사항을 보여주었습니다. 그런 다음 영웅 목록과 영웅 세부 정보가 모두 포함 된 마스터 / 세부 폼이 되었습니다. 곧 새로운 요구 사항과 기능이 생길 것입니다. 한 구성 요소에 계속해서 기능을 추가할 수는 없습니다. 그것은 유지되기 어렵습니다.

특정 작업이나 워크 플로에 중점을 둔 하위 컴포넌트로 기능을 분리해야합니다. 결국, AppComponent는 이러한 하위 컴포넌트를 소유하는 단순한 상자가 될 수 있습니다.

이 페이지에서는 영웅의 세부 사항을 별도의 재사용 가능한 구성 요소로 분리함으로써 해당 방향의 첫 번째 단계를 수행합니다.

```
ng g component heroDetail --flat
```

### hero-detail.component.ts

```
import { Component, OnInit, Input } from '@angular/core';
import { Hero } from './hero';

@Component({
  selector: 'hero-detail',
  templateUrl: './hero-detail.component.html',
  styleUrls: ['./hero-detail.component.css']
})
export class HeroDetailComponent implements OnInit {
  @Input() hero: Hero;

  constructor() { }

  ngOnInit() {
  }
}
```

### hero-detail.component.html

```
<div *ngIf="hero">
  <h2>{{hero.name}} details!</h2>
</div>
```

```
<label>id: </label>{{hero.id}}
</div>
<div>
  <label>name: </label>
  <input [(ngModel)]="hero.name" placeholder="name"/>
</div>
</div>
```

```
cd src/app && ng g class hero
```

### hero.ts

```
export class Hero {
  id: number;
  name: string;
}
```

### app.component.html

```
<h1>{{title}}</h1>
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes" (click)="onSelect(hero)" [class.selected]="hero === selectedHero">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>
<!-- <div *ngIf="selectedHero">
  <h2>{{selectedHero.name}} details!</h2>
  <div><label>id: </label>{{selectedHero.id}}</div>
  <div>
    <label>name: </label>
    <input [(ngModel)]="selectedHero.name" placeholder="name"/>
  </div>
</div> -->
<hero-detail [hero]="selectedHero"></hero-detail>
```

# Tour of Heroes

## My Heroes

11	Mr. Nice
12	Narco
13	Bombasto
14	Celeritas
15	Magneta
16	RubberMan
17	Dynama
18	Dr IQ
19	Magma
20	Tornado

## Mr. Nice details!

id: 11

name:

## app.component.ts

```
import { Component } from '@angular/core';
import { Hero } from './hero';
// export class Hero {
//   id: number;
//   name: string;
// }

const HEROES: Hero[] = [
  { id: 11, name: 'Mr. Nice' },
  { id: 12, name: 'Narco' },
  { id: 13, name: 'Bombasto' },
  { id: 14, name: 'Celeritas' },
  { id: 15, name: 'Magneta' },
  { id: 16, name: 'RubberMan' },
  { id: 17, name: 'Dynama' },
  { id: 18, name: 'Dr IQ' },
  { id: 19, name: 'Magma' },
  { id: 20, name: 'Tornado' }
];
```

```
@Component({
```

```

    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Tour of Heroes';
  // hero: Hero = {
  //   id: 1,
  //   name: 'Windstorm'
  // };
  heroes = HEROES;

  selectedHero: Hero;
  onSelect(hero: Hero): void {
    this.selectedHero = hero;
  }
}

```

### app.module.ts

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';

import { FormsModule } from '@angular/forms'; // <-- NgModel lives here

import { AppComponent } from './app.component';
import { HeroDetailComponent } from './hero-detail.component';

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule // <-- import the FormsModule before binding with [(ngModel)]
  ],
  declarations: [
    AppComponent,
    HeroDetailComponent
  ],
  providers: [],
  bootstrap: [AppComponent]
})

```

```
export class AppModule { }
```

## 04 : Services

Tour of Heroes 앱이 진화하면서 영웅 데이터에 액세스해야하는 더 많은 컴포넌트가 추가됩니다.

동일한 코드를 반복해서 복사하여 붙여 넣는 대신 재사용 가능한 단일 데이터 서비스를 만들어 필요한 컴포넌트에 삽입합니다. 별도의 서비스를 사용하면 컴포넌트를 간결하게 유지하고 뷰 지원에 주력 할 수 있으며 모의 서비스로 컴포넌트 테스트를 쉽게 수행 할 수 있습니다.

데이터 서비스는 언제나 비동기식이므로 Promise 기반 버전의 데이터 서비스로 페이지를 마칩니다.

### Creating a hero service

이해 관계자들은 다양한 페이지에서 영웅을 다양한 방식으로 보여주기를 원합니다. 사용자는 이미 목록에서 영웅을 선택할 수 있습니다. 조만간 최고 실적 영웅들과 함께 다시 보드를 추가하고 영웅 세부 사항을 편집 할 수 있는 별도의 화면을 만듭니다. 세 가지 화면 모두 영웅 데이터가 필요합니다.

현재 AppComponent는 모의 영웅을 정의하여 표시합니다. 그러나 영웅을 정의하는 것은 컴포넌트의 일이 아니므로 영웅 목록을 다른 컴포넌트 및 화면과 쉽게 공유 할 수 없습니다. 이 페이지에서 영웅 데이터 수집 비즈니스 로직을 데이터를 제공하는 단일 서비스로 옮기고 해당 서비스를 데이터가 필요한 모든 컴포넌트와 공유합니다.

#### mock-heroes.ts

```
import { Hero } from './hero';

export const HEROES: Hero[] = [
  { id: 11, name: 'Mr. Nice' },
  { id: 12, name: 'Narco' },
  { id: 13, name: 'Bombasto' },
  { id: 14, name: 'Celeritas' },
  { id: 15, name: 'Magneta' },
  { id: 16, name: 'RubberMan' },
  { id: 17, name: 'Dynamite' },
  { id: 18, name: 'Dr IQ' },
  { id: 19, name: 'Magma' },
  { id: 20, name: 'Tornado' }
];
```

```
cd src/app && ng g service hero
```

### hero.service.ts

```
import { Injectable } from '@angular/core';
import { Hero } from './hero';
import { HEROES } from './mock-heroes';

@Injectable()
export class HeroService {

  constructor() { }

  // getHeroes(): Hero[] {
  //   return HEROES;
  // }

  getHeroes(): Promise<Hero[]> {
    return Promise.resolve(HEROES);
  }

}
```

### app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';

import { FormsModule } from '@angular/forms'; // <-- NgModel lives here

import { AppComponent } from './app.component';
import { HeroDetailComponent } from './hero-detail.component';

import { HeroService } from './hero.service';

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule // <-- import the FormsModule before binding with [(ngModel)]
  ],
})
```

```

declarations: [
  AppComponent,
  HeroDetailComponent
],
providers: [HeroService],
bootstrap: [AppComponent]
})
export class AppModule { }

```

### app.component.ts

```

import { Component, OnInit } from '@angular/core';
import { Hero } from './hero';
import { HeroService } from './hero.service';

// const HEROES: Hero[] = [
//   { id: 11, name: 'Mr. Nice' },
//   { id: 12, name: 'Narco' },
//   { id: 13, name: 'Bombasto' },
//   { id: 14, name: 'Celeritas' },
//   { id: 15, name: 'Magneta' },
//   { id: 16, name: 'RubberMan' },
//   { id: 17, name: 'Dynamite' },
//   { id: 18, name: 'Dr IQ' },
//   { id: 19, name: 'Magma' },
//   { id: 20, name: 'Tornado' }
// ];

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'Tour of Heroes';
  // heroes = HEROES;
  heroes: Hero[];
  selectedHero: Hero;

  constructor(private heroService: HeroService) {}

  ngOnInit(): void {

```

```

    this.getHeroes());
}

onSelect(hero: Hero): void {
  this.selectedHero = hero;
}

// getHeroes(): void {
//   this.heroes = this.heroService.getHeroes();
// }

getHeroes(): void {
  this.heroService.getHeroes().then(heroes => this.heroes = heroes);
}

}

```

## Tour of Heroes

### My Heroes

11	Mr. Nice
12	Narco
13	Bombasto
14	Celeritas
15	Magneta
16	RubberMan
17	Dynama
18	Dr IQ
19	Magma
20	Tornado

### Dynama details!

id: 17

name:

### hero.service.ts

```

import { Injectable } from '@angular/core';
import { Hero } from './hero';

```

```

import { HEROES } from './mock-heroes';

@Injectable()
export class HeroService {

  constructor() { }

  getHeroes(): Promise<Hero[]> {
    return Promise.resolve(HEROES);
  }

  getHeroesSlowly(): Promise<Hero[]> {
    return new Promise(resolve => {
      // Simulate server latency with 2 second delay
      setTimeout(() => resolve(this.getHeroes()), 2000);
    });
  }
}

```

### app.component.ts

```

import { Component, OnInit } from '@angular/core';
import { Hero } from './hero';
import { HeroService } from './hero.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'Tour of Heroes';
  heroes: Hero[];
  selectedHero: Hero;

  constructor(private heroService: HeroService) {}

  ngOnInit(): void {
    this.getHeroes();
  }

  getHeroes(): void {
    this.heroService.getHeroes().subscribe(heroes => {
      this.heroes = heroes;
    });
  }

  onSelect(hero: Hero): void {
    this.selectedHero = hero;
  }
}

```

```
onSelect(hero: Hero): void {
    this.selectedHero = hero;
}

getHeroes(): void {
    // this.heroService.getHeroes().then(heroes => this.heroes = heroes);
    this.heroService.getHeroesSlowly().then(heroes => this.heroes = heroes);
}

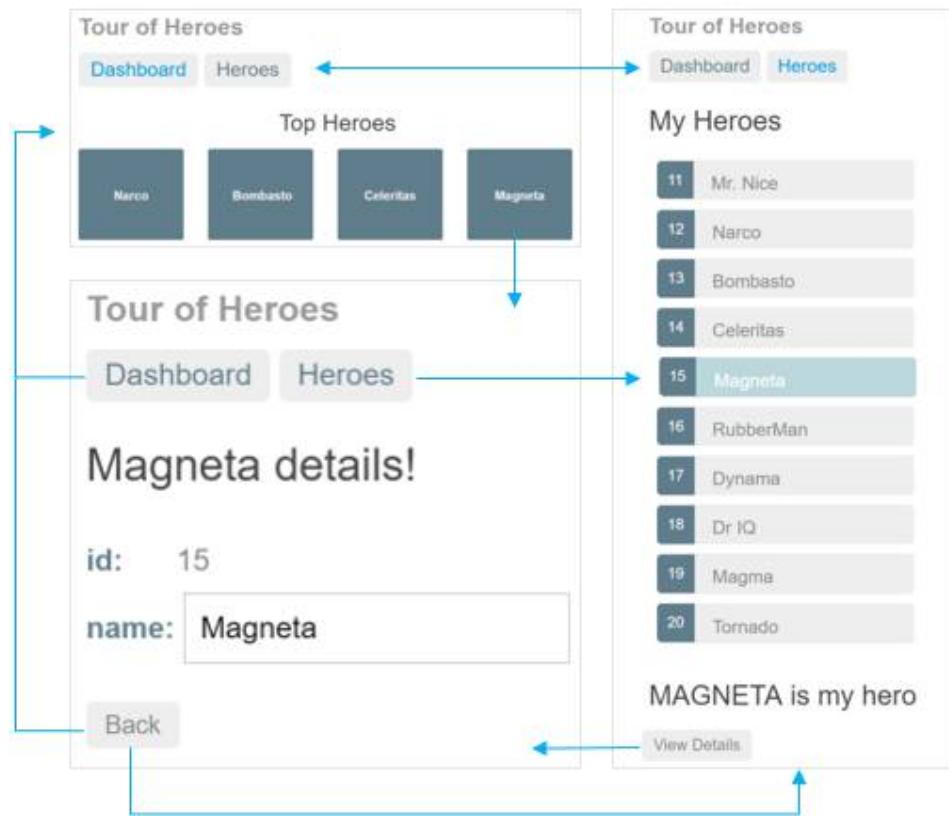
}
```

## 05 : Routing

Tour of Heroes 앱에는 새로운 요구 사항이 있습니다.

- 대시 보드 뷰를 추가하십시오.
- 영웅 뷰와 대시 보드 뷰간에 이동하는 기능을 추가하십시오.
- 사용자가 어느 뷰에서 영웅 이름을 클릭하면 선택한 영웅의 상세 뷰로 이동합니다.
- 사용자가 이메일의 링크를 클릭하면 특정 영웅에 대한 세부 정보 뷰를 엽니다.

완료되면 사용자가 다음과 같이 앱을 탐색 할 수 있습니다.



계획은 다음과 같습니다.

- AppComponent는 네비게이션만 처리하는 셀역할을 수행하도록 합니다.
- 현재 AppComponent 내의 Heroes 관심사를 별도의 HeroesComponent로 재배치하십시오.
- 라우팅을 추가하십시오.
- 새 DashboardComponent를 만듭니다.
- 대시 보드를 네비게이션 구조에 연결하십시오.

```
ng g component heroes --flat
```

### heroes.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'my-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

```
ng g component dashboard --flat
```

### dashboard.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Hero } from './hero';
import { HeroService } from './hero.service';

@Component({
  selector: 'my-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
  heroes: Hero[] = [];

  constructor(private heroService: HeroService) {}

  ngOnInit(): void {
    this.heroService.getHeroes()
  }
}
```

```

        .then(heroes => this.heroes = heroes.slice(1, 5));
    }
}

```

### **dashboard.component.html**

```

<h3>Top Heroes</h3>
<div class="grid grid-pad">
  <div *ngFor="let hero of heroes" class="col-1-4">
    <div class="module hero">
      <h4>{{hero.name}}</h4>
    </div>
  </div>
</div>

```

### **app.component.html**

```

<!-- <h1>{{title}}</h1>
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes" (click)="onSelect(hero)" [class.selected]="hero === selectedHero">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>
<hero-detail [hero]="selectedHero"></hero-detail> -->
<h1>{{title}}</h1>
<nav>
  <a routerLink="/dashboard">Dashboard</a>
  <a routerLink="/heroes">Heroes</a>
</nav>
<router-outlet></router-outlet>

```

### **app-routing.module.ts**

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HeroesComponent } from './heroes.component';
import { DashboardComponent } from './dashboard.component';

const routes: Routes = [
  { path: 'dashboard', component: DashboardComponent },

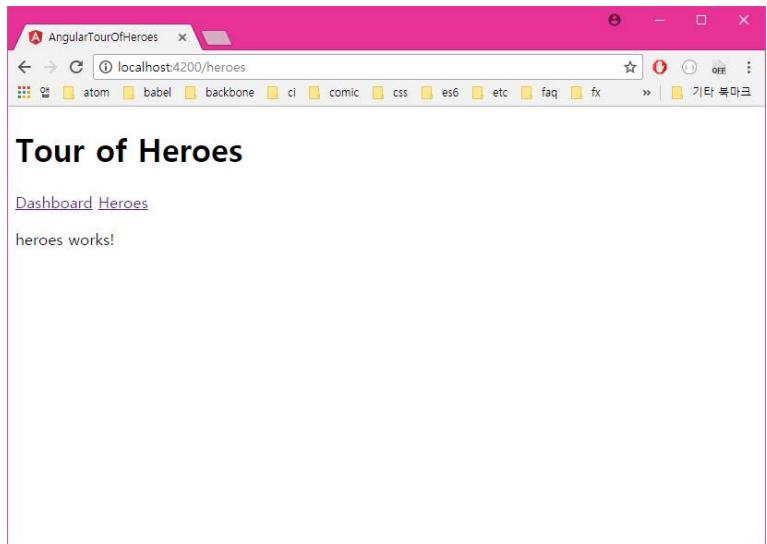
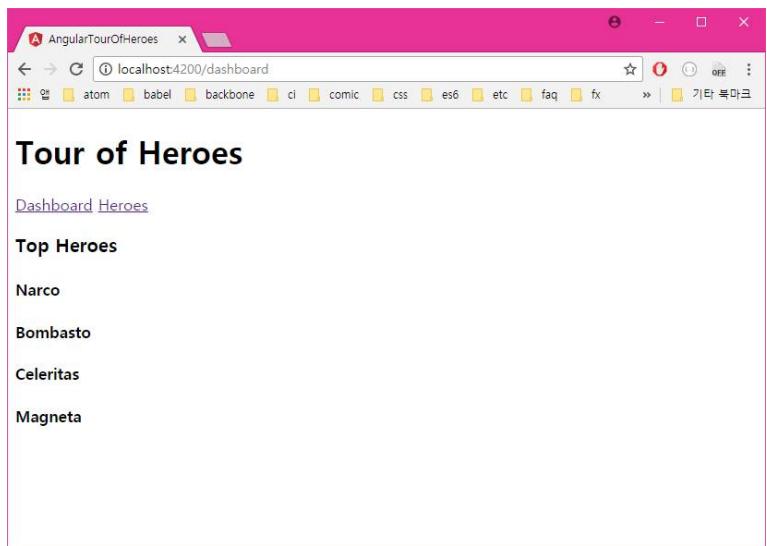
```

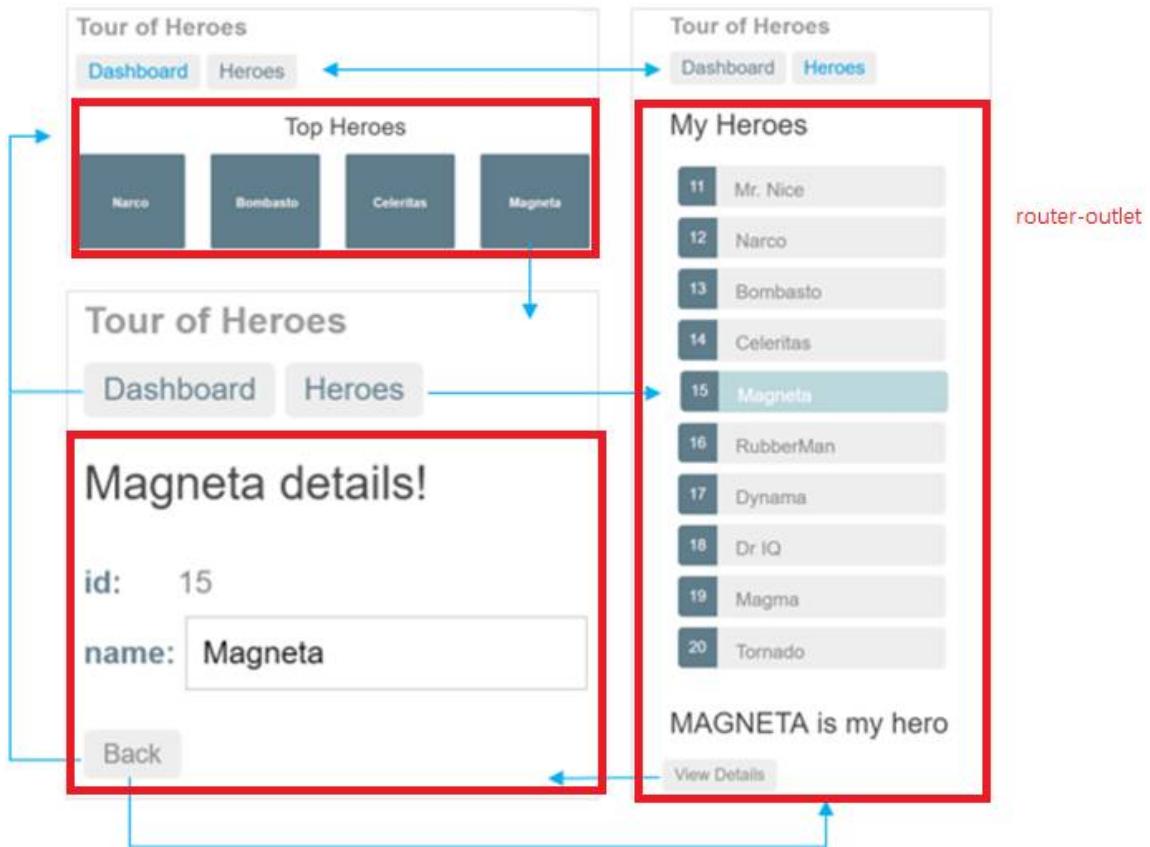
```

    { path: 'heroes', component: HeroesComponent },
    { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```





## app.module.ts

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';

import { FormsModule } from '@angular/forms';
import { RouterModule } from '@angular/router';

import { AppComponent } from './app.component';
import { DashboardComponent } from './dashboard.component';
import { HeroDetailComponent } from './hero-detail.component';
import { HeroesComponent } from './heroes.component';

import { HeroService } from './hero.service';

```

```

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  declarations: [
    AppComponent,
    DashboardComponent,
    HeroDetailComponent,
    HeroesComponent,
  ],
  providers: [HeroService],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

### hero.service.ts

```

import { Injectable } from '@angular/core';
import { Hero } from './hero';
import { HEROES } from './mock-heroes';

@Injectable()
export class HeroService {

  getHeroes(): Promise<Hero[]> {
    return Promise.resolve(HEROES);
  }

  getHeroesSlowly(): Promise<Hero[]> {
    return new Promise(resolve => {
      // Simulate server latency with 2 second delay
      setTimeout(() => resolve(this.getHeroes()), 2000);
    });
  }

  getHero(id: number): Promise<Hero> {
    return this.getHeroes()
      .then(heroes => heroes.find(hero => hero.id === id));
  }
}

```

```
}
```

### app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { DashboardComponent } from './dashboard.component';
import { HeroesComponent } from './heroes.component';
import { HeroDetailComponent } from './hero-detail.component';

const routes: Routes = [
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'detail/:id', component: HeroDetailComponent },
  { path: 'heroes', component: HeroesComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

### app.component.ts

```
import { Component, OnInit } from '@angular/core';
// import { Hero } from './hero';
// import { HeroService } from './hero.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'Tour of Heroes';
  // heroes: Hero[];
  // selectedHero: Hero;

  // constructor(private heroService: HeroService) { }
```

```

ngOnInit(): void {
    // this.getHeroes();
}

// onSelect(hero: Hero): void {
//     this.selectedHero = hero;
// }

// getHeroes(): void {
//     this.heroService.getHeroes().then(heroes => this.heroes = heroes);
//     this.heroService.getHeroesSlowly().then(heroes => this.heroes = heroes);
// }

}

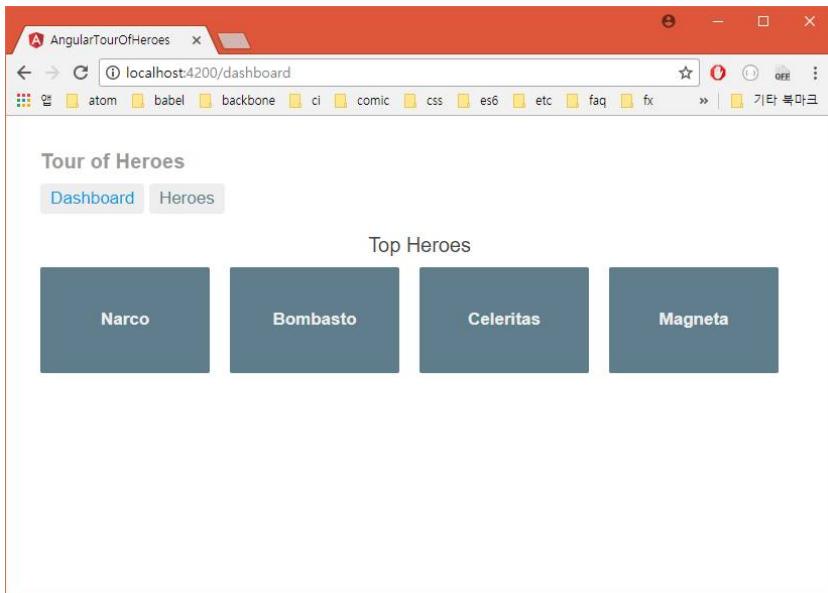
```

### app.component.html

```

<!-- <h1>{{title}}</h1>
<h2>My Heroes</h2>
<ul class="heroes">
    <li *ngFor="let hero of heroes" (click)="onSelect(hero)" [class.selected]="hero === selectedHero">
        <span class="badge">{{hero.id}}</span> {{hero.name}}
    </li>
</ul>
<hero-detail [hero]="selectedHero"></hero-detail> -->
<h1>{{title}}</h1>
<nav>
    <a routerLink="/dashboard" routerLinkActive="active">Dashboard</a>
    <a routerLink="/heroes" routerLinkActive="active">Heroes</a>
</nav>
<router-outlet></router-outlet>

```



## dashboard.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Hero } from './hero';
import { HeroService } from './hero.service';

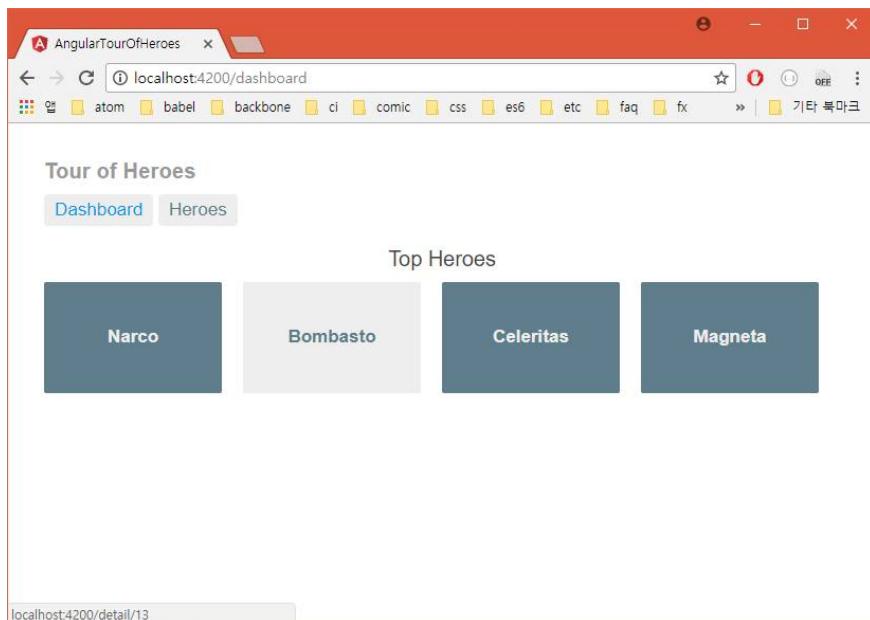
@Component({
  selector: 'my-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
  heroes: Hero[] = [];

  constructor(private heroService: HeroService) { }

  ngOnInit(): void {
    this.heroService.getHeroes()
      .then(heroes => this.heroes = heroes.slice(1, 5));
  }
}
```

## dashboard.component.html

```
<h3>Top Heroes</h3>
<div class="grid grid-pad">
  <!-- <div *ngFor="let hero of heroes" class="col-1-4">
    <div class="module hero">
      <h4>{{hero.name}}</h4>
    </div>
  </div> -->
  <a *ngFor="let hero of heroes" [routerLink]="['/detail', hero.id]" class="col-1-4">
    <div class="module hero">
      <h4>{{hero.name}}</h4>
    </div>
  </a>
</div>
```



## hero-detail.component.ts

```
import { Component, OnInit, Input } from '@angular/core';
import { Hero } from './hero';
import { ActivatedRoute, ParamMap } from '@angular/router';
import { Location } from '@angular/common';
import { HeroService } from './hero.service';
import 'rxjs/add/operator/switchMap';

@Component({
```

```

selector: 'hero-detail',
templateUrl: './hero-detail.component.html',
styleUrls: ['./hero-detail.component.css']
})

export class HeroDetailComponent implements OnInit {
  // @Input()
  hero: Hero;

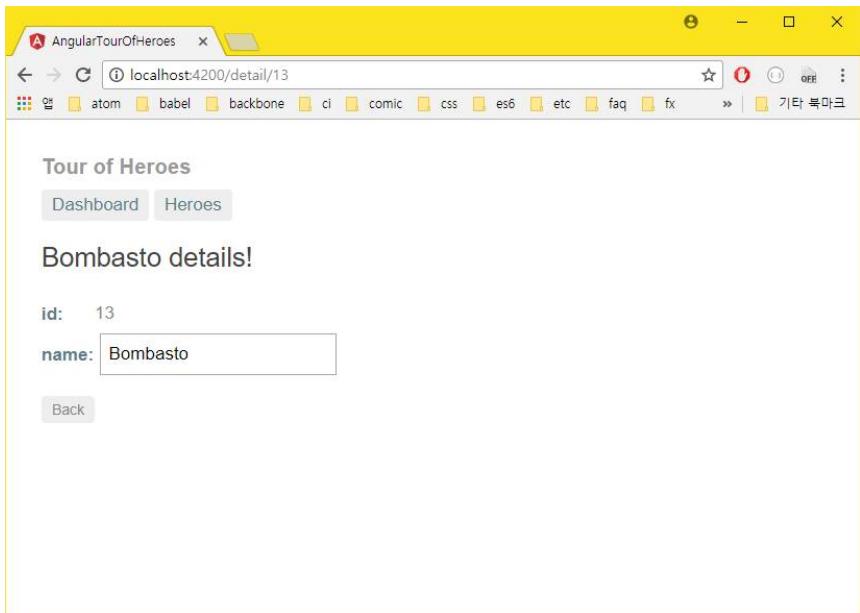
  constructor(
    private heroService: HeroService,
    private route: ActivatedRoute,
    private location: Location
  ) { }

  ngOnInit(): void {
    this.route.paramMap
      .switchMap((params: ParamMap) => this.heroService.getHero(+params.get('id')))
      .subscribe(hero => this.hero = hero);
  }

  goBack(): void {
    this.location.back();
  }
}

```

- getHero 요청이 처리되는 동안 사용자가 이 구성 요소를 다시 탐색(링크를 클릭)하면 switchMap 은 이전 요청을 취소 한 다음 HeroService.getHero ()를 다시 호출합니다.
- 영웅 ID 는 숫자입니다. 경로 매개 변수는 항상 문자열입니다. 따라서 route 매개 변수 값은 JavaScript (+) 연산자를 사용하여 숫자로 변환됩니다.



## heroes.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';

import { Hero } from './hero';
import { HeroService } from './hero.service';

@Component({
  selector: 'my-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;

  constructor(
    private router: Router,
    private heroService: HeroService) {}

  ngOnInit(): void {
    this.getHeroes();
  }

  getHeroes(): void {
```

```

    this.heroService.getHeroes().then(heroes => this.heroes = heroes);
}

onSelect(hero: Hero): void {
  this.selectedHero = hero;
}

gotoDetail(): void {
  this.router.navigate(['/detail', this.selectedHero.id]);
}

}

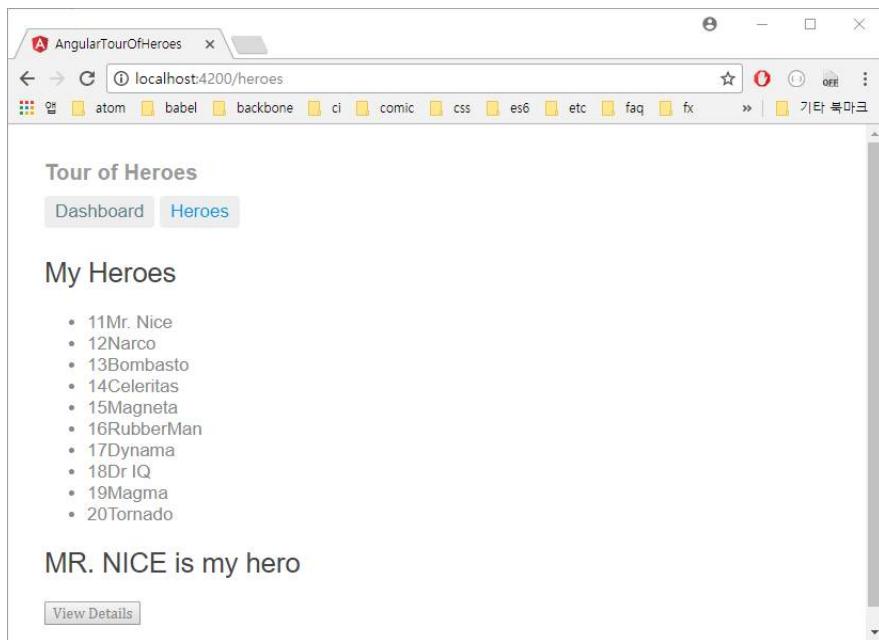
```

### **heroes.component.html**

```

<h1>{{title}}</h1>
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes" [class.selected]="hero === selectedHero" (click)="onSelect(hero)">
    <span class="badge">{{hero.id}}</span>{{hero.name}}
  </li>
</ul>
<!-- <hero-detail [hero]="selectedHero"></hero-detail> -->
<div *ngIf="selectedHero">
  <h2>
    {{selectedHero.name | uppercase}} is my hero
  </h2>
  <button (click)="gotoDetail()">View Details</button>
</div>

```



## styles.css

```
/* Master Styles */

h1 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}

h2, h3 {
  color: #444;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: lighter;
}

body {
  margin: 2em;
}

body, input[text], button {
  color: #888;
  font-family: Cambria, Georgia;
}
```

```
/* everywhere else */

* {
  font-family: Arial, Helvetica, sans-serif;
}
```

## app.component.css

```
h1 {
  font-size: 1.2em;
  color: #999;
  margin-bottom: 0;
}

h2 {
  font-size: 2em;
  margin-top: 0;
  padding-top: 0;
}

nav a {
  padding: 5px 10px;
  text-decoration: none;
  margin-top: 10px;
  display: inline-block;
  background-color: #eee;
  border-radius: 4px;
}

nav a:visited, a:link {
  color: #607D8B;
}

nav a:hover {
  color: #039be5;
  background-color: #CFD8DC;
}

nav a.active {
  color: #039be5;
}
```

## dashboard.component.css

```
[class*='col-'] {  
    float: left;  
    padding-right: 20px;  
    padding-bottom: 20px;  
}  
  
[class*='col-']:last-of-type {  
    padding-right: 0;  
}  
  
a {  
    text-decoration: none;  
}  
  
*, *:after, *:before {  
    -webkit-box-sizing: border-box;  
    -moz-box-sizing: border-box;  
    box-sizing: border-box;  
}  
  
h3 {  
    text-align: center;  
    margin-bottom: 0;  
}  
  
h4 {  
    position: relative;  
}  
  
.grid {  
    margin: 0;  
}  
  
.col-1-4 {  
    width: 25%;  
}  
  
.module {  
    padding: 20px;  
    text-align: center;  
    color: #eee;
```

```
max-height: 120px;
min-width: 120px;
background-color: #607D8B;
border-radius: 2px;
}

.module:hover {
  background-color: #EEE;
  cursor: pointer;
  color: #607d8b;
}

.grid-pad {
  padding: 10px 0;
}

.grid-pad>[class*='col-']:last-of-type {
  padding-right: 20px;
}

@media (max-width: 600px) {
  .module {
    font-size: 10px;
    max-height: 75px;
  }
}

@media (max-width: 1024px) {
  .grid {
    margin: 0;
  }
  .module {
    min-width: 60px;
  }
}
```

### hero-detail.component.css

```
label {
  display: inline-block;
  width: 3em;
  margin: .5em 0;
```

```
color: #607D8B;
font-weight: bold;
}

input {
  height: 2em;
  font-size: 1em;
  padding-left: .4em;
}

button {
  margin-top: 20px;
  font-family: Arial;
  background-color: #eee;
  border: none;
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer;
  cursor: hand;
}

button:hover {
  background-color: #cf8dc;
}

button:disabled {
  background-color: #eee;
  color: #ccc;
  cursor: auto;
}
```

### heroes.component.css

```
.selected {
  background-color: #CFD8DC !important;
  color: white;
}

.heroes {
  margin: 0 0 2em 0;
  list-style-type: none;
  padding: 0;
```

```
width: 15em;
}

.heroes li {
  cursor: pointer;
  position: relative;
  left: 0;
  background-color: #EEE;
  margin: .5em;
  padding: .3em 0;
  height: 1.6em;
  border-radius: 4px;
}

.heroes li:hover {
  color: #607D8B;
  background-color: #DDD;
  left: .1em;
}

.heroes li.selected:hover {
  background-color: #BBB8DC !important;
  color: white;
}

.heroes .text {
  position: relative;
  top: -3px;
}

.heroes .badge {
  display: inline-block;
  font-size: small;
  color: white;
  padding: 0.8em 0.7em 0 0.7em;
  background-color: #607D8B;
  line-height: 1em;
  position: relative;
  left: -1px;
  top: -4px;
  height: 1.8em;
  margin-right: .8em;
  border-radius: 4px 0 0 4px;
```

```
}
```

```
button {
```

```
    font-family: Arial;
```

```
    background-color: #eee;
```

```
    border: none;
```

```
    padding: 5px 10px;
```

```
    border-radius: 4px;
```

```
    cursor: pointer;
```

```
    cursor: hand;
```

```
}
```

```
button:hover {
```

```
    background-color: #cf8dc;
```

```
}
```

## 06 : Http

이번 작업에서 다음과 같이 개선 할 것입니다.

- 서버에서 영웅 데이터를 가져옵니다.
- 사용자가 영웅 이름을 추가, 편집 및 삭제할 수 있습니다.

HttpModule은 코어 앱러모듈이 아닙니다. HttpModule은 웹 액세스에 대한 Angular의 선택적인 접근 방식입니다. @angular/http라고하는 별도의 애드온 모듈로 존재하며 Angular npm 패키지의 일부로 별도의 스크립트 파일로 제공됩니다.

```
npm install angular-in-memory-web-api --save
```

### app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';

import { FormsModule }   from '@angular/forms';
import { HttpClientModule }    from '@angular/http';

// Imports for loading & configuring the in-memory web api
import { InMemoryWebApiModule } from 'angular-in-memory-web-api';
import { InMemoryDataService }  from './in-memory-data.service';

import { AppComponent } from './app.component';
import { DashboardComponent } from './dashboard.component';
import { HeroDetailComponent } from './hero-detail.component';
import { HeroesComponent } from './heroes.component';

import { HeroService } from './hero.service';

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule,
    InMemoryWebApiModule.forRoot(InMemoryDataService),
  ],
})
```

```

declarations: [
  AppComponent,
  DashboardComponent,
  HeroDetailComponent,
  HeroesComponent,
],
providers: [HeroService],
bootstrap: [AppComponent]
})
export class AppModule { }

```

### in-memory-data.service.ts

```

import { InMemoryDbService } from 'angular-in-memory-web-api';

export class InMemoryDataService implements InMemoryDbService {
  createDb() {
    const heroes = [
      { id: 0, name: 'Zero' },
      { id: 11, name: 'Mr. Nice' },
      { id: 12, name: 'Narco' },
      { id: 13, name: 'Bombasto' },
      { id: 14, name: 'Celeritas' },
      { id: 15, name: 'Magneta' },
      { id: 16, name: 'RubberMan' },
      { id: 17, name: 'Dynamite' },
      { id: 18, name: 'Dr IQ' },
      { id: 19, name: 'Magma' },
      { id: 20, name: 'Tornado' }
    ];
    return { heroes };
  }
}

```

### hero.service.ts

```

import { Injectable } from '@angular/core';
import { Hero } from './hero';
// import { HEROES } from './mock-heroes';
import { Headers, Http } from '@angular/http';
import 'rxjs/add/operator/toPromise';

```

```

@Injectable()
export class HeroService {
  private heroesUrl = 'api/heroes'; // URL to web api

  constructor(private http: Http) { }

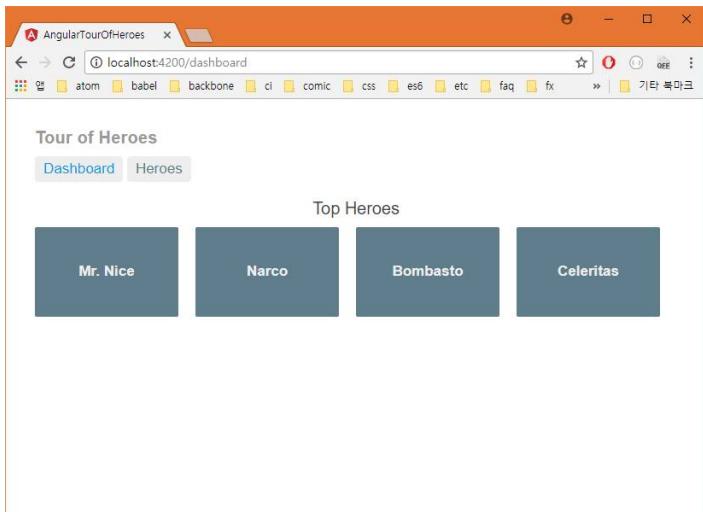
  getHeroes(): Promise<Hero[]> {
    // return Promise.resolve(HEROES);
    return this.http.get(this.heroesUrl)
      .toPromise()
      .then(response => response.json().data as Hero[])
      .catch(this.handleError);
  }

  private handleError(error: any): Promise<any> {
    console.error('An error occurred', error); // for demo purposes only
    return Promise.reject(error.message || error);
  }

  getHeroesSlowly(): Promise<Hero[]> {
    return new Promise(resolve => {
      setTimeout(() => resolve(this.getHeroes()), 2000);
    });
  }

  getHero(id: number): Promise<Hero> {
    return this.getHeroes()
      .then(heroes => heroes.find(hero => hero.id === id));
  }
}

```



Angular http.get은 RxJS Observable을 반환합니다. toPromise 연산자를 사용하여 Observable을 Promise로 변환했습니다. Angular Observable에는 toPromise 연산자가 기본적으로 제공되지 않습니다.

ObPvable을 유용한 기능으로 확장하는 toPromise와 같은 연산자가 많이 있습니다. 이러한 기능을 사용하려면 연산자 자체를 추가해야합니다. RxJS 라이브러리에서 다음과 같이 쉽게 가져올 수 있습니다.

```
import 'rxjs/add/operator/toPromise';
```

### mock-heroes.ts

```
// import { Hero } from './hero';
//
// export const HEROES: Hero[] = [
//   { id: 11, name: 'Mr. Nice' },
//   { id: 12, name: 'Narco' },
//   { id: 13, name: 'Bombasto' },
//   { id: 14, name: 'Celeritas' },
//   { id: 15, name: 'Magneta' },
//   { id: 16, name: 'RubberMan' },
//   { id: 17, name: 'Dynamna' },
//   { id: 18, name: 'Dr IQ' },
//   { id: 19, name: 'Magma' },
//   { id: 20, name: 'Tornado' }
//];
```

## hero.service.ts

```
import { Injectable } from '@angular/core';
import { Hero } from './hero';
// import { HEROES } from './mock-heroes';
import { Headers, Http } from '@angular/http';
import 'rxjs/add/operator/toPromise';

@Injectable()
export class HeroService {
  private heroesUrl = 'api/heroes'; // URL to web api

  constructor(private http: Http) { }

  getHeroes(): Promise<Hero[]> {
    // return Promise.resolve(HEROES);
    return this.http.get(this.heroesUrl)
      .toPromise()
      .then(response => response.json().data as Hero[])
      .catch(this.handleError);
  }

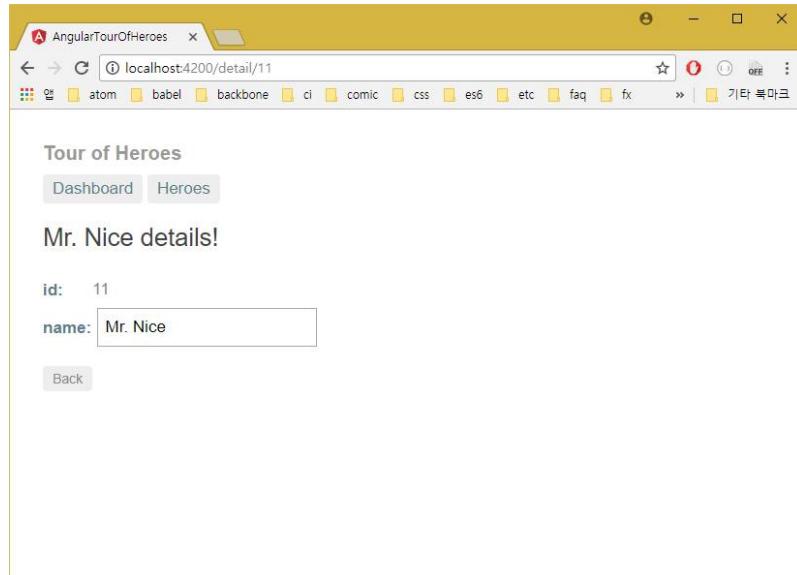
  private handleError(error: any): Promise<any> {
    console.error('An error occurred', error); // for demo purposes only
    return Promise.reject(error.message || error);
  }

  getHeroesSlowly(): Promise<Hero[]> {
    return new Promise(resolve => {
      setTimeout(() => resolve(this.getHeroes()), 2000);
    });
  }

  getHero(id: number): Promise<Hero> {
    // return this.getHeroes()
    //   .then(heroes => heroes.find(hero => hero.id === id));
    const url = `${this.heroesUrl}/${id}`;
    return this.http.get(url)
      .toPromise()
      .then(response => response.json().data as Hero)
      .catch(this.handleError);
  }
}
```

}

HeroDetailComponent가 HeroService에게 영웅을 가져 오도록 요청하면 HeroService는 현재 일치하는 ID를 가진 영웅과 필터를 모두 가져옵니다. 시뮬레이션에는 문제가 없지만 실제 서버에 모든 영웅을 물어 보는 것은 낭비입니다. 대부분의 웹 API는 api / hero / : id 형식 (api / hero / 11 등)으로 get-by-id 요청을 지원합니다.



## 수정기능 추가

### hero-detail.component.html

```
<div *ngIf="hero">
  <h2>{{hero.name}} details! </h2>
  <div>
    <label>id: </label>{{hero.id}}
  </div>
  <div>
    <label>name: </label>
    <input [(ngModel)]="hero.name" placeholder="name"/>
  </div>
  <button (click)="goBack()">Back</button>
  <button (click)="save()">Save</button>
</div>
```

### hero-detail.component.ts

```
import { Component, OnInit, Input } from '@angular/core';
import { Hero } from './hero';
import { ActivatedRoute, ParamMap } from '@angular/router';
import { Location } from '@angular/common';
import { HeroService } from './hero.service';
import 'rxjs/add/operator/switchMap';

@Component({
  selector: 'hero-detail',
  templateUrl: './hero-detail.component.html',
  styleUrls: ['./hero-detail.component.css']
})
export class HeroDetailComponent implements OnInit {
  hero: Hero;

  constructor(
    private heroService: HeroService,
    private route: ActivatedRoute,
    private location: Location
  ) {}
```

```

ngOnInit(): void {
  this.route.paramMap
    .switchMap((params: ParamMap) => this.heroService.getHero(+params.get('id')))
    .subscribe(hero => this.hero = hero);
}

goBack(): void {
  this.location.back();
}

save(): void {
  this.heroService.update(this.hero)
    .then(() => this.goBack());
}

}

```

### hero.service.ts

```

import { Injectable } from '@angular/core';
import { Hero } from './hero';
// import { HEROES } from './mock-heroes';
import { Headers, Http } from '@angular/http';
import 'rxjs/add/operator/toPromise';

@Injectable()
export class HeroService {
  private heroesUrl = 'api/heroes'; // URL to web api
  private headers = new Headers({'Content-Type': 'application/json'});

  constructor(private http: Http) {}

  getHeroes(): Promise<Hero[]> {
    // return Promise.resolve(HEROES);
    return this.http.get(this.heroesUrl)
      .toPromise()
      .then(response => response.json().data as Hero[])
      .catch(this.handleError);
  }

  private handleError(error: any): Promise<any> {
    console.error('An error occurred', error); // for demo purposes only
  }
}

```

```

        return Promise.reject(error.message || error);
    }

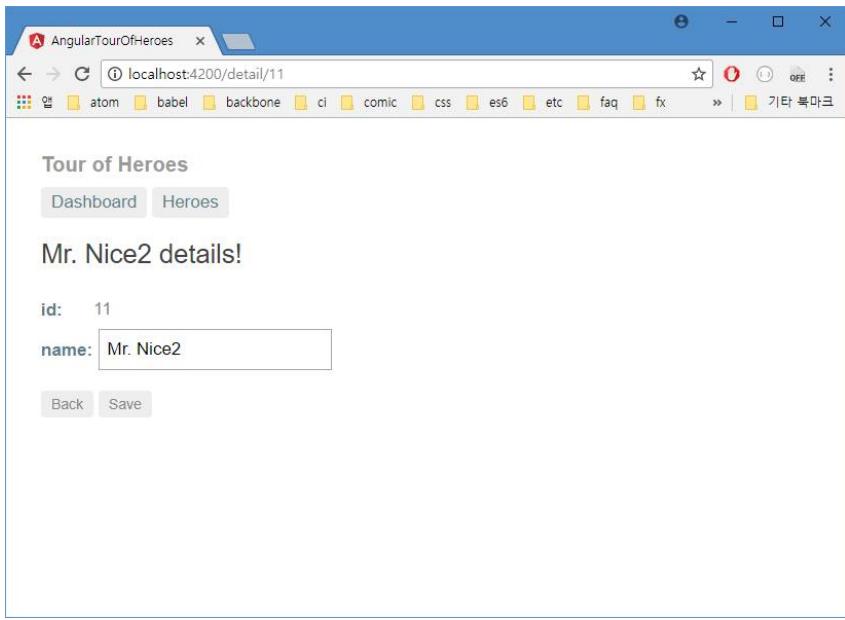
getHeroesSlowly(): Promise<Hero[]> {
    return new Promise(resolve => {
        setTimeout(() => resolve(this.getHeroes()), 2000);
    });
}

getHero(id: number): Promise<Hero> {
    // return this.getHeroes()
    //     .then(heroes => heroes.find(hero => hero.id === id));
    const url = `${this.heroesUrl}/${id}`;
    return this.http.get(url)
        .toPromise()
        .then(response => response.json().data as Hero)
        .catch(this.handleError);
}

update(hero: Hero): Promise<Hero> {
    const url = `${this.heroesUrl}/${hero.id}`;
    return this.http
        .put(url, JSON.stringify(hero), { headers: this.headers })
        .toPromise()
        .then(() => hero)
        .catch(this.handleError);
}

}

```



## 입력기능 추가

### heroes.component.html

```
<h2>My Heroes</h2>
<div>
  <label>Hero name:</label> <input #heroName />
  <button (click)="add(heroName.value); heroName.value=''">Add</button>
</div>
<ul class="heroes">
  <li *ngFor="let hero of heroes" [class.selected]="hero === selectedHero" (click)="onSelect(hero)">
    <span class="badge">{{hero.id}}</span>{{hero.name}}
  </li>
</ul>
<div *ngIf="selectedHero">
  <h2>
    {{selectedHero.name | uppercase}} is my hero
  </h2>
  <button (click)="gotoDetail()">View Details</button>
</div>
```

### heroes.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';

import { Hero } from './hero';
import { HeroService } from './hero.service';

@Component({
  selector: 'my-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;

  constructor(
    private router: Router,
    private heroService: HeroService) {
```

```

        console.log('HeroesComponent#constructor() called.');
    }

    ngOnInit(): void {
        this.getHeroes();
    }

    getHeroes(): void {
        this.heroService.getHeroes().then(heroes => this.heroes = heroes);
    }

    onSelect(hero: Hero): void {
        this.selectedHero = hero;
    }

    gotoDetail(): void {
        this.router.navigate(['/detail', this.selectedHero.id]);
    }

    add(name: string): void {
        name = name.trim();
        if(!name) {
            return;
        }
        this.heroService.create(name)
            .then(hero => {
                this.heroes.push(hero);
                this.selectedHero = null;
            });
    }
}

```

## hero.service.ts

```

import { Injectable } from '@angular/core';
import { Hero } from './hero';
import { Headers, Http } from '@angular/http';
import 'rxjs/add/operator/toPromise';

@Injectable()
export class HeroService {
    private heroesUrl = 'api/heroes';

```

```

private headers = new Headers({'Content-Type': 'application/json'});

constructor(private http: Http) { }

getHeroes(): Promise<Hero[]> {
  return this.http.get(this.heroesUrl)
    .toPromise()
    .then(response => response.json().data as Hero[])
    .catch(this.handleError);
}

private handleError(error: any): Promise<any> {
  console.error('An error occurred', error);
  return Promise.reject(error.message || error);
}

getHeroesSlowly(): Promise<Hero[]> {
  return new Promise(resolve => {
    setTimeout(() => resolve(this.getHeroes()), 2000);
  });
}

getHero(id: number): Promise<Hero> {
  const url = `${this.heroesUrl}/${id}`;
  return this.http.get(url)
    .toPromise()
    .then(response => response.json().data as Hero)
    .catch(this.handleError);
}

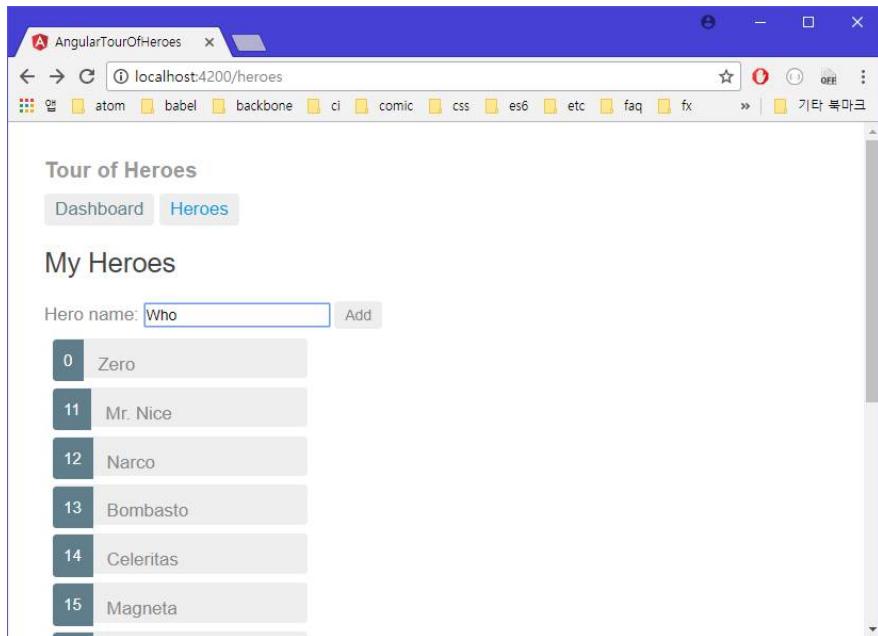
update(hero: Hero): Promise<Hero> {
  const url = `${this.heroesUrl}/${hero.id}`;
  return this.http
    .put(url, JSON.stringify(hero), { headers: this.headers })
    .toPromise()
    .then(() => hero)
    .catch(this.handleError);
}

create(name: string): Promise<Hero> {
  return this.http
    .post(this.heroesUrl, JSON.stringify({name: name}), {headers: this.headers})
    .toPromise()
}

```

```
.then(res => res.json().data as Hero)
      .catch(this.handleError);
}

}
```



## 삭제기능 추가

### heroes.component.html

```
<h2>My Heroes</h2>
<div>
  <label>Hero name:</label> <input #heroName />
  <button (click)="add(heroName.value); heroName.value=''">Add</button>
</div>
<ul class="heroes">
  <li *ngFor="let hero of heroes" [class.selected]="hero === selectedHero" (click)="onSelect(hero)">
    <span class="badge">{{hero.id}}</span>
    <span>{{hero.name}}</span>
    <button class="delete" (click)="delete(hero); $event.stopPropagation()">x</button>
  </li>
</ul>
<div *ngIf="selectedHero">
  <h2>
    {{selectedHero.name | uppercase}} is my hero
  </h2>
  <button (click)="gotoDetail()">View Details</button>
</div>
```

### heroes.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';

import { Hero } from './hero';
import { HeroService } from './hero.service';

@Component({
  selector: 'my-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;

  constructor()
```

```

private router: Router,
private heroService: HeroService) {
  console.log('HeroesComponent#constructor() called.');
}

ngOnInit(): void {
  this.getHeroes();
}

getHeroes(): void {
  this.heroService.getHeroes().then(heroes => this.heroes = heroes);
}

onSelect(hero: Hero): void {
  this.selectedHero = hero;
}

gotoDetail(): void {
  this.router.navigate(['/detail', this.selectedHero.id]);
}

add(name: string): void {
  name = name.trim();
  if(!name) {
    return;
  }
  this.heroService.create(name)
    .then(hero => {
      this.heroes.push(hero);
      this.selectedHero = null;
    });
}

delete (hero: Hero): void {
  this.heroService
    .delete(hero.id)
    .then(() => {
      this.heroes = this.heroes.filter(h => h !== hero);
      if (this.selectedHero === hero) { this.selectedHero = null; }
    });
}
}

```

## hero.service.ts

```
import { Injectable } from '@angular/core';
import { Hero } from './hero';
import { Headers, Http } from '@angular/http';
import 'rxjs/add/operator/toPromise';

@Injectable()
export class HeroService {
  private heroesUrl = 'api/heroes';
  private headers = new Headers({ 'Content-Type': 'application/json' });

  constructor(private http: Http) { }

  getHeroes(): Promise<Hero[]> {
    return this.http.get(this.heroesUrl)
      .toPromise()
      .then(response => response.json().data as Hero[])
      .catch(this.handleError);
  }

  private handleError(error: any): Promise<any> {
    console.error('An error occurred', error);
    return Promise.reject(error.message || error);
  }

  getHeroesSlowly(): Promise<Hero[]> {
    return new Promise(resolve => {
      setTimeout(() => resolve(this.getHeroes()), 2000);
    });
  }

  getHero(id: number): Promise<Hero> {
    const url = `${this.heroesUrl}/${id}`;
    return this.http.get(url)
      .toPromise()
      .then(response => response.json().data as Hero)
      .catch(this.handleError);
  }

  update(hero: Hero): Promise<Hero> {
    const url = `${this.heroesUrl}/${hero.id}`;
    return this.http
```

```

    .put(url, JSON.stringify(hero), { headers: this.headers })
    .toPromise()
    .then(() => hero)
    .catch(this.handleError);
}

create(name: string): Promise<Hero> {
  return this.http
    .post(this.heroesUrl, JSON.stringify({name: name}), {headers: this.headers})
    .toPromise()
    .then(res => res.json().data as Hero)
    .catch(this.handleError);
}

delete (id: number): Promise<void> {
  const url = `${this.heroesUrl}/${id}`;
  return this.http.delete(url, { headers: this.headers })
    .toPromise()
    .then(() => null)
    .catch(this.handleError);
}
}

```

### heroes.component.css 파일에 다음을 추가

```

button.delete {
  float:right;
  margin-top: 2px;
  margin-right: .8em;
  background-color: gray !important;
  color:white;
}

```

The screenshot shows a web browser window with the title "AngularTourOfHeroes". The address bar displays "localhost:4200/heroes". The page content is titled "Tour of Heroes" and shows the "Heroes" tab selected. Below the tabs, the heading "My Heroes" is displayed. A search input field labeled "Hero name:" is followed by a "Add" button. A list of heroes is shown in a table format:

	Hero Name	Action
0	Zero	X
12	Narco	X
13	Bombasto	X
14	Celeritas	X
15	Magneta	X
16	RubberMan	X

## 검색기능 추가

Observable은 배열과 비슷한 연산자들을 지원하는 이벤트의 스트림입니다. 앵귤러 코어는 Observable을 기본적으로 지원합니다. 개발자는 RxJS 라이브러리를 사용하여 지원을 확대할수 있습니다.

HeroService는 toPromise 연산자를 http.get()의 관찰 가능한 결과에 연결시켰습니다. 그 연산자는 Observable을 Promise로 변환했고 그 약속을 호출자에게 다시 전달했습니다.

Promise로 전환하는 것은 좋은 선택입니다. 일반적으로 http.get()에 단일 데이터 청크를 가져 오도록 요청합니다. 데이터를 받으면 작업이 완료됩니다. 호출 구성 요소는 약속의 형태로 단일 결과를 쉽게 소비 할 수 있습니다.

그러나 요청이 항상 한 번만 수행되는 것은 아닙니다. 서버가 첫 번째 요청에 응답하기 전에 하나의 요청을 시작하여 취소하고 다른 요청을 할 수 있습니다.

요청-취소-새 요청 시퀀스는 Promise으로 구현하기는 어렵지만 Observables에서는 쉽게 구현할 수 있습니다.

```
cd src/app && ng g service hero-search
```

### hero-search.service.ts

```
import { Injectable } from '@angular/core';
import { Http }      from '@angular/http';

import { Observable }    from 'rxjs/Observable';
import 'rxjs/add/operator/map';

import { Hero }          from './hero';

@Injectable()
export class HeroSearchService {

  constructor(private http: Http) { }

  search(term: string): Observable<Hero[]> {
    return this.http
      .get(`api/heroes/?name=${term}`)
      .map(response => response.json().data as Hero[]);
  }
}
```

```
ng g component hero-search --flat
```

### hero-search.component.html

```
<div id="search-component">
  <h4>Hero Search</h4>
  <input #searchBox id="search-box" (keyup)="search(searchBox.value)"/>
  <div>
    <div *ngFor="let hero of heroes | async" (click)="gotoDetail(hero)" class="search-result">
      {{hero.name}}
    </div>
  </div>
</div>
```

### hero-search.component.css

```
.search-result {
  border-bottom: 1px solid gray;
  border-left: 1px solid gray;
  border-right: 1px solid gray;
  width: 195px;
  height: 16px;
  padding: 5px;
  background-color: white;
  cursor: pointer;
}

.search-result:hover {
  color: #eee;
  background-color: #607D8B;
}

#search-box {
  width: 200px;
  height: 20px;
}
```

### hero-search.component.ts

```
import { Component, OnInit } from '@angular/core';
```

```

import { Router }           from '@angular/router';
import { Observable }       from 'rxjs/Observable';
import { Subject }          from 'rxjs/Subject';

// Observable class extensions
import 'rxjs/add/observable/of';

// Observable operators
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/debounceTime';
import 'rxjs/add/operator/distinctUntilChanged';

import { HeroSearchService } from './hero-search.service';
import { Hero } from './hero';

@Component({
  selector: 'hero-search',
  templateUrl: './hero-search.component.html',
  styleUrls: ['./hero-search.component.css']
})
export class HeroSearchComponent implements OnInit {
  heroes: Observable<Hero[]>;
  private searchTerms = new Subject<string>();

  constructor(
    private heroSearchService: HeroSearchService,
    private router: Router) { }

  // Push a search term into the observable stream.
  search(term: string): void {
    this.searchTerms.next(term);
  }

  ngOnInit(): void {
    this.heroes = this.searchTerms
      .debounceTime(300)           // wait 300ms after each keystroke before considering the term
      .distinctUntilChanged()     // ignore if next search term is same as previous
      .switchMap(term => term)   // switch to new observable each time the term changes
        // return the http search observable
      ? this.heroSearchService.search(term)
        // or the observable of empty heroes if there was no search term
      : Observable.of<Hero[]>([])
  }
}

```

```

    .catch(error => {
      // TODO: add real error handling
      console.log(error);
      return Observable.of<Hero[]>([]);
    });
}

gotoDetail(hero: Hero): void {
  let link = ['/detail', hero.id];
  this.router.navigate(link);
}

}

```

Subject는 관찰 가능한(Observable) 이벤트 스트림의 생성자입니다. searchTerms는 이름 검색을 위한 필터 기준 인 Observable 문자열을 생성합니다. search ()를 호출 할 때마다 next ()를 호출하여 이 주제의 관찰 가능한 스트림에 새로운 문자열을 넣습니다.

```

private searchTerms = new Subject<string>();

// Push a search term into the observable stream.
search(term: string): void {
  this.searchTerms.next(term);
}

```

Subject는 Observable입니다. 검색 용어의 스트림을 Hero 배열의 스트림으로 변환하고 그 결과를 heroes 속성에 저장할 수 있습니다.

모든 사용자 키 스트로크를 직접 HeroSearchService에 전달하면 과도한 양의 HTTP 요청이 만들어져 서버 리소스에 부담이됩니다. 대신, Observable 연산자를 연결하여 Observable 문자열에 대한 요청 흐름을 줄일 수 있습니다. HeroSearchService에 대한 호출 횟수를 줄여서 적절한 결과를 얻습니다.

```

heroes: Observable<Hero[]>;

ngOnInit(): void {
  this.heroes = this.searchTerms
    .debounceTime(300)          // wait 300ms after each keystroke before considering the term
    .distinctUntilChanged()     // ignore if next search term is same as previous
    .switchMap(term => term    // switch to new observable each time the term changes
      // return the http search observable
      ? this.heroSearchService.search(term)
      // or the observable of empty heroes if there was no search term
      : Observable.of<Hero[]>([]))
}

```

```

    .catch(error => {
      // TODO: add real error handling
      console.log(error);
      return Observable.of<Hero[]>([[]]);
    });
}

```

debounceTime (300)은 새로운 문자열 이벤트의 흐름이 300 밀리 초 동안 일시 중지 될 때까지 기다린 후 최신 문자열을 전달합니다. 300ms보다 자주 요청을하지 않습니다.

distinctUntilChanged는 필터 텍스트가 변경된 경우에만 요청을 보냅니다.

switchMap ()은 debounce 및 distinctUntilChanged를 통해 검색하는 각 검색어에 대해 검색 서비스를 호출합니다. 이전 검색 관측 값을 취소하고 파기하며 관찰 가능한 최신 검색 서비스 만 반환합니다.

```

import { Observable }      from 'rxjs/Observable';
import { Subject }        from 'rxjs/Subject';

// Observable class extensions
import 'rxjs/add/observable/of';

// Observable operators
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/debounceTime';
import 'rxjs/add/operator/distinctUntilChanged';

```

대부분의 RxJS 연산자는 Angular의 Observable 기본 구현에 포함되어 있지 않습니다. 기본 구현에는 Angular 자체에서 필요한 것만 포함됩니다.

더 많은 RxJS 기능이 필요하면 정의 된 라이브러리를 가져 와서 Observable을 확장하십시오.

라이브러리를 가져 오는 단순한 작업만으로 라이브러리의 스크립트 파일을로드하고 실행하며, 그러면 해당 연산자가 Observable 클래스에 추가됩니다.

## app.module.ts

```

import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';

import { FormsModule }   from '@angular/forms';
import { HttpClientModule } from '@angular/http';

```

```

// Imports for loading & configuring the in-memory web api
import { InMemoryWebApiModule } from 'angular-in-memory-web-api';
import { InMemoryDataService } from './in-memory-data.service';

import { AppComponent } from './app.component';
import { DashboardComponent } from './dashboard.component';
import { HeroDetailComponent } from './hero-detail.component';
import { HeroesComponent } from './heroes.component';
import { HeroSearchComponent } from './hero-search.component';

import { HeroService } from './hero.service';
import { HeroSearchService } from './hero-search.service';

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpModule,
    InMemoryWebApiModule.forRoot(InMemoryDataService),
  ],
  declarations: [
    AppComponent,
    DashboardComponent,
    HeroDetailComponent,
    HeroesComponent,
    HeroSearchComponent,
  ],
  providers: [HeroService, HeroSearchService],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

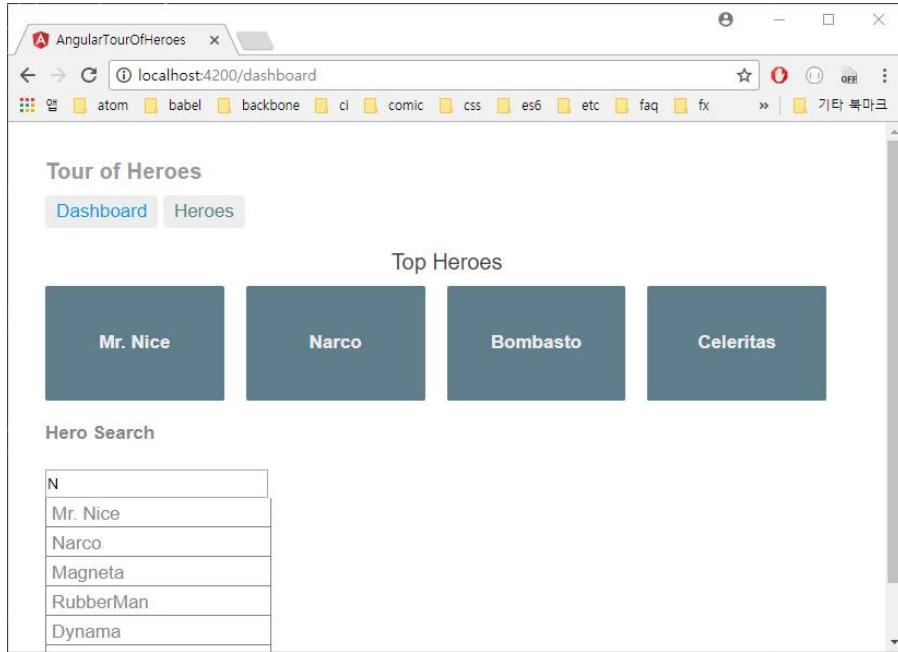
## dashboard.component.html

```

<h3>Top Heroes</h3>
<div class="grid grid-pad">
  <a *ngFor="let hero of heroes" [routerLink]="['/detail', hero.id]" class="col-1-4">
    <div class="module hero">
      <h4>{{hero.name}}</h4>
    </div>
  </a>
</div>

```

```
</div>
</a>
</div>
<hero-search></hero-search>
```



## Angular.js

*Developed by Google. First released in 2009*

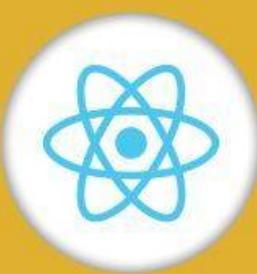
- Popular among developers
- Low cost of first UI version (C)
- A lot of tools that are perfect for teamwork
- Good for quick and dirty solutions
- Better fit for enterprise-grade applications than React
- Base packages are maintained by Google



*Developed by Facebook.*

*First released as open source in 2013 under BSD license.*

## React.js



- Easier to scale
- Predictable states (cheaper scale)
- Good for big front-end projects
- Relatively small API
- Constant re-rendering of components provides efficient arranging at increasing complexity

*Polymer was released by Google back in 2013.*

## Polymer.js



- Quick speed
- Opportunity to create custom elements
- Provide templating and bi-directional data binding
- Reduces the gap between developer and designer
- Good for feature-rich applications

## Vue.js

*Developed by Evan You. Released in 2014*

- Very simple API
- Allows to sue selective modules
- Consistently adoptable
- Easy to integrate with other libraries or existing projects
- Updates model and view via two-way data binding
- Good for large scale applications





## Chapter 6 : Etc

알아두면 도움이 되는 기능을 추가적으로 살펴본다.

# Transclusion using ng-content

```
import { Component, OnInit } from '@angular/core';

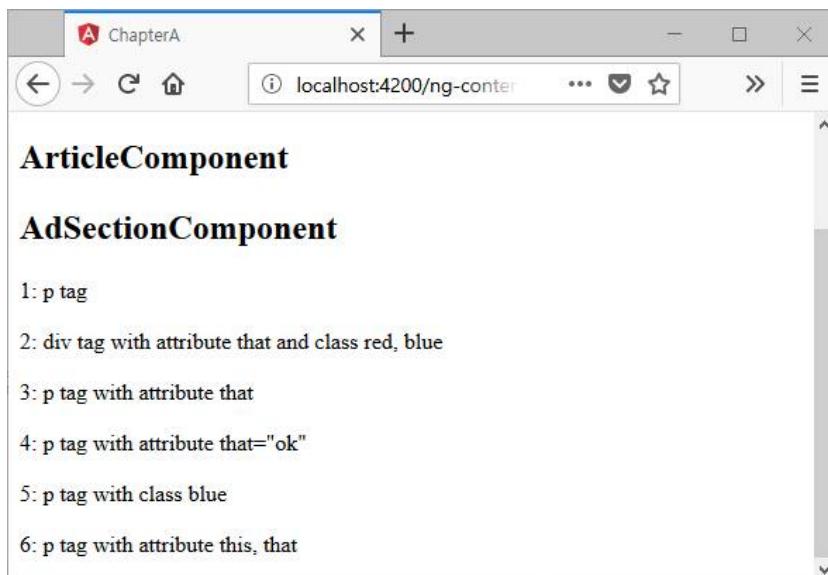
@Component({
  selector: 'article',
  template: `
    <h1>{{title}}</h1>
    <ad-section>
      <p>1: p tag</p>
      <div that class="red blue">2: div tag with attribute that and class red, blue</div>
      <p that>3: p tag with attribute that</p>
      <p that="ok">4: p tag with attribute that="ok"</p>
      <p class="blue">5: p tag with class blue</p>
      <p this that>6: p tag with attribute this, that</p>
    </ad-section>
  `,
  styleUrls: ['./article.component.css']
})
export class ArticleComponent implements OnInit {
  title = 'ArticleComponent';
  constructor() { }
  ngOnInit() { }
}
```

ArticleComponent 컴포넌트가 ad-section 셀렉터를 사용하면서 다수의 태그를 ad-section 셀렉터가 가리키는 AdSectionComponent 컴포넌트의 자식으로 배정하고 있습니다.

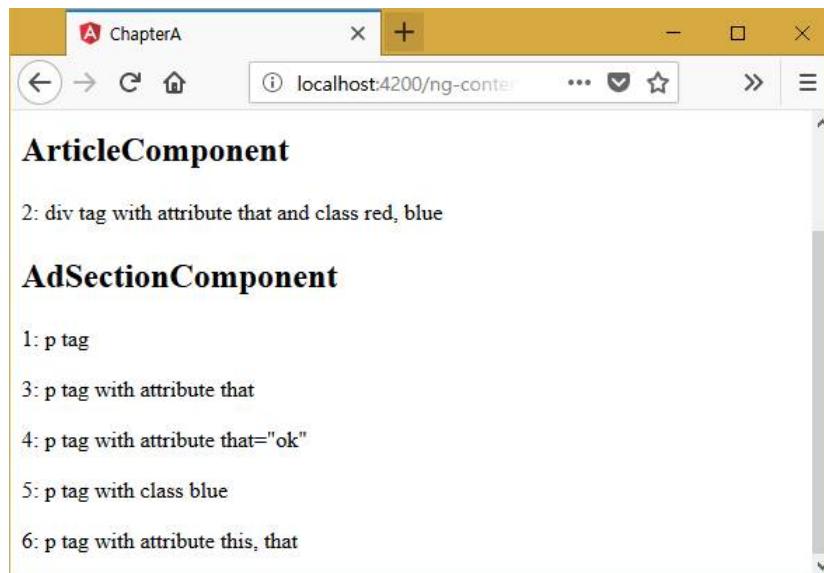
AdSectionComponent 컴포넌트는 이렇게 배정된 엘리먼트들의 배치위치를 지정할 수 있습니다.

```
import { Component, OnInit } from '@angular/core';

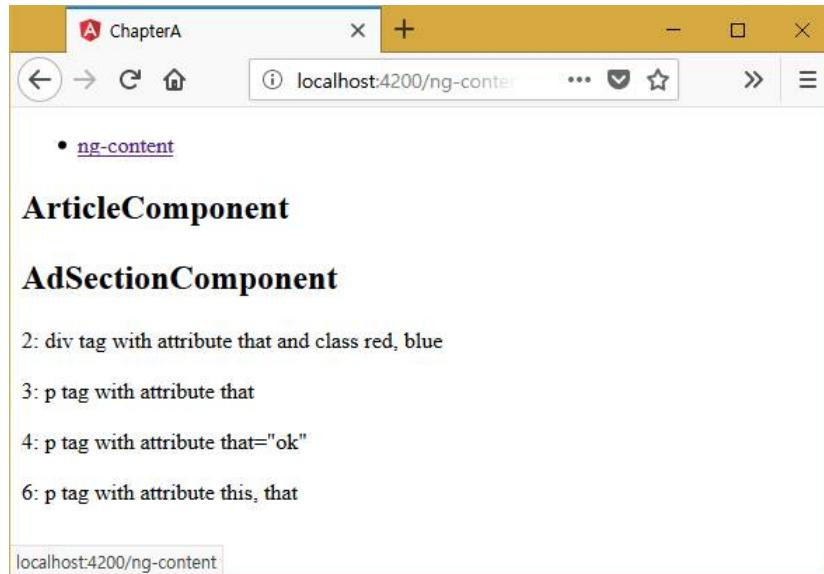
@Component({
  selector: 'ad-section',
  // 부모 컴포넌트가 추가하여 현 컴포넌트의 자식이 된 엘리먼트의 배치 위치를 지정한다.
  template: `
    <h1>{{title}}</h1>
    <ng-content></ng-content>
  `,
  styleUrls: ['./ad-section.component.css']
})
export class AdSectionComponent implements OnInit {
  title = 'AdSectionComponent';
  constructor() { }
  ngOnInit() { }
}
```



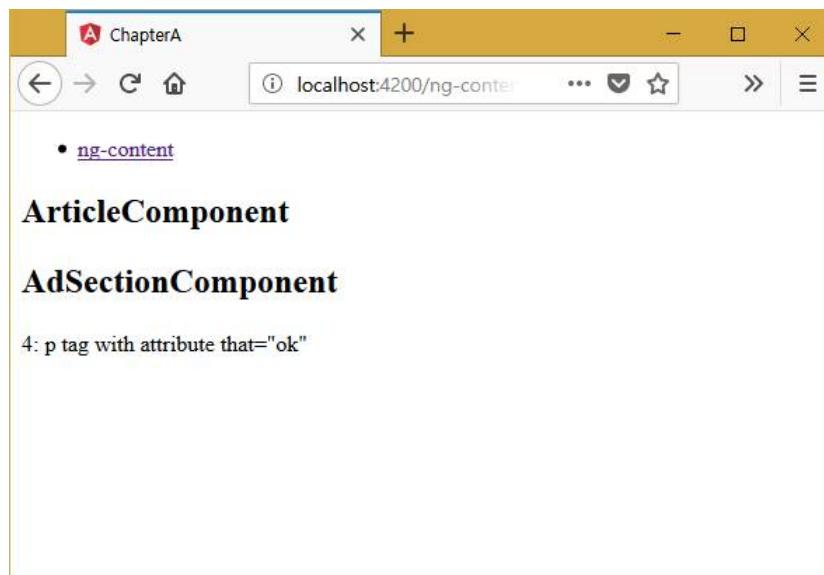
```
// 부모 컴포넌트가 추가한 엘리먼트를 태그로 필터링하여 구분한 다음 위치를 지정한다.  
template: `  
  <ng-content select="div"></ng-content>  
  <h1>{{title}}</h1>  
  <ng-content select="p"></ng-content>  
`  
,
```



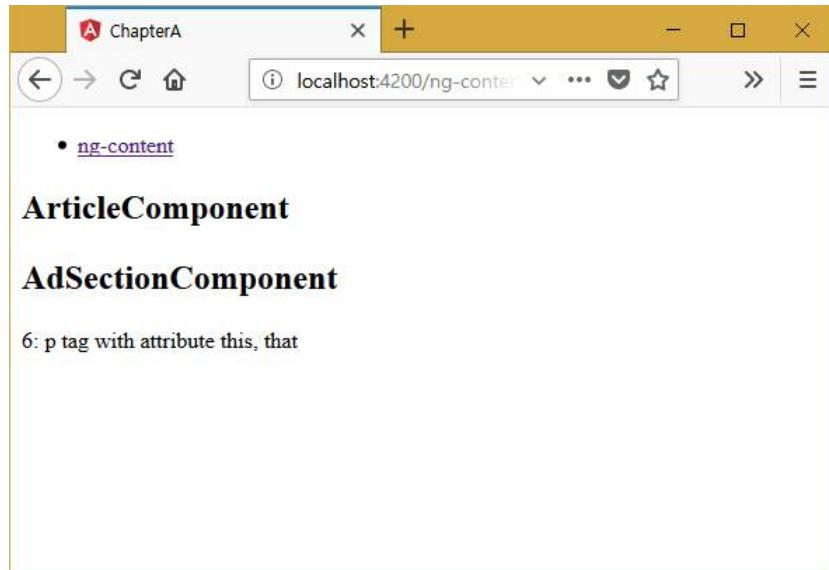
```
// 부모 컴포넌트가 추가한 엘리먼트를 해당 속성의 존재여부로 필터링하여 구분한 다음 위치를 지정한다.  
template: `  
  <h1>{{title}}</h1>  
  <ng-content select="[that]"></ng-content>  
`
```



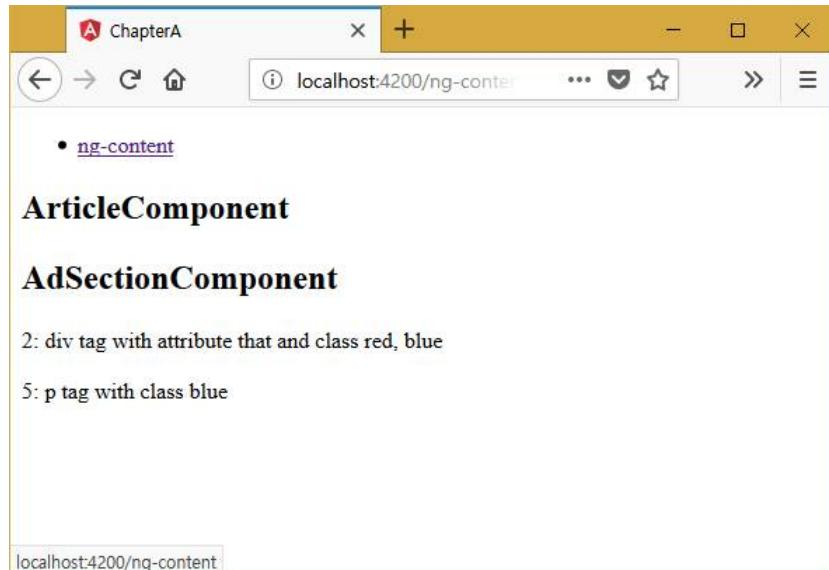
```
// 부모 컴포넌트가 추가한 엘리먼트를 해당 속성값 설정의 존재여부로 필터링하여
// 구분한 다음 위치를 지정한다.
template: `
  <h1>{{title}}</h1>
  <ng-content select="[that=ok]"></ng-content>
`,
```



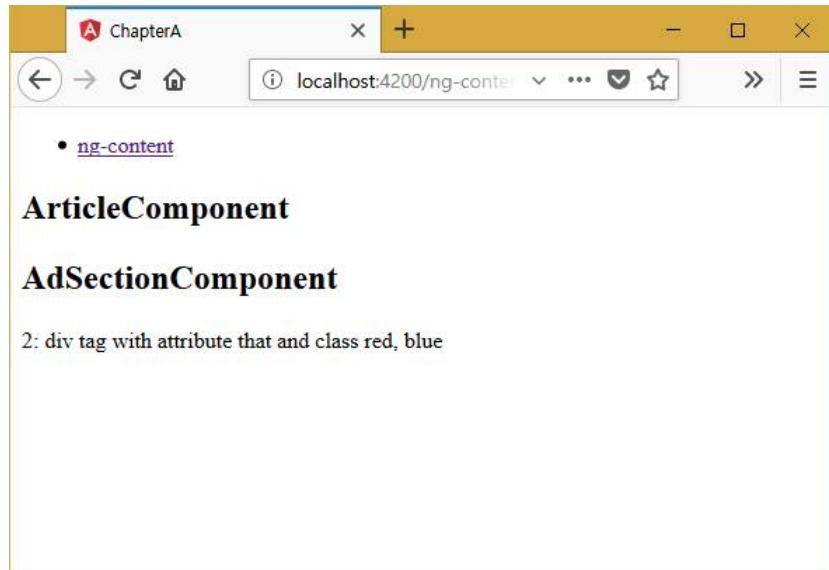
```
// 부모 컴포넌트가 추가한 엘리먼트를 해당 속성들의 존재여부로 필터링하여 구분한 다음 위치를 지정한다.  
template: `  
  <h1>{{title}}</h1>  
  <ng-content select="[that][this]"></ng-content>  
`
```



```
// 부모 컴포넌트가 추가한 엘리먼트를 해당 클래스의 존재여부로 필터링하여 구분한 다음 위치를 지정한다.  
template: `  
  <h1>{{title}}</h1>  
  <ng-content select=".blue"></ng-content>  
`
```



```
// 부모 컴포넌트가 추가한 엘리먼트를 해당 클래스들의 존재여부로 필터링하여 구분한 다음 위치를 지정한다.  
template: `  
  <h1>{{title}}</h1>  
  <ng-content select=".red.blue"></ng-content>  
`
```



## Shadow DOM

컴포넌트 자신이 직접 선택하여 사용하는 모든 로컬 DOM 엘리먼트입니다. 여기에는 모든 하위 컴포넌트들도 포함됩니다.

```
@Component({
  selector: 'song-track',
  template: `
    <track-title>{{track}}</track-title>
    <track-artist>{{artist}}</track-artist>`
})
export class SongTrack { }
```

```
<song-track title="No Lie" artist="Sean Paul..."></song-track>
```

## Light DOM

부모 컴포넌트가 지정해준 모든 하위 DOM 엘리먼트입니다. 투영 된 콘텐츠(ng-content)라고도 합니다.

앞서서 살펴 본 Transclusion 파트를 참고하세요.

```
@Component({
  selector: 'song-track',
  template: `<ng-content></ng-content>`
})
export class SongTrack { }
```

```
<song-track>
  <track-title>No Lie</track-title>
  <track-artist>Sean Paul, Dua Lipa</track-artist>
</song-track>
```

# ViewEncapsulation

## \_nghost-\*, \_ngcontent-\*

앵귤러가 사용하는 \_nghost-\*, \_ngcontent-\* 설정 및 효력범위에 대해서 살펴봅니다.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
    <style media="screen">
      [_nghost-1] h1 {
        color: red;
      }

      h2[_ngcontent-1]{
        color: blue;
      }

      h3 [_ngcontent-2]{
        color: green;
      }

      .selected {
        color: orange;
      }
    </style>
  </head>
  <body>
    <div _nghost-1>
      <h1>RED</h1>
    </div>
    <h1>NOT RED</h1>

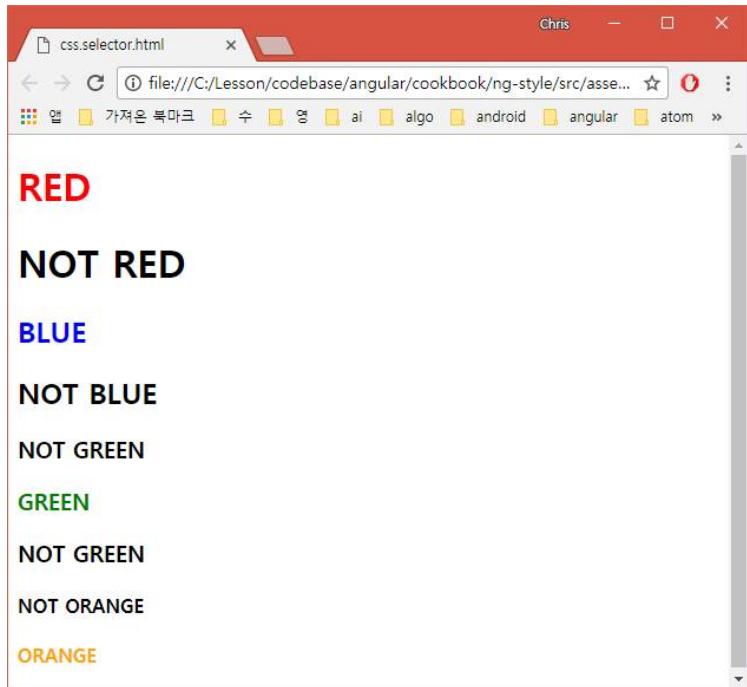
    <div>
      <h2 _ngcontent-1>BLUE</h2>
    </div>
    <h2>NOT BLUE</h2>

    <div>
      <h3 _ngcontent-2>NOT GREEN</h3>
      <h3><label _ngcontent-2>GREEN</label></h3>
    </div>
    <h3>NOT GREEN</h3>

    <child>
      <h4>NOT ORANGE</h4>
    </child>
    <child class="selected">
      <h4>ORANGE</h4>
    </child>
  </body>
</html>
```

child 를 셀렉터로 사용하는 ChildComponent 컴포넌트의 CSS 설정에서 사용하는 :host 셀렉터 또는 컴포넌트 클래

스에서 사용하는 @HostBinding 데코레이터는 ChildComponent 컴포넌트 영역을 의미한다.



## ViewEncapsulation.Emulated

ViewEncapsulation.Emulated 설정은 기본값이다. 따라서, 이 설정을 하지 않아도 이 설정을 명시적으로 한 것과 같다.

### app.component.ts

```
import { Component, ViewEncapsulation } from '@angular/core';

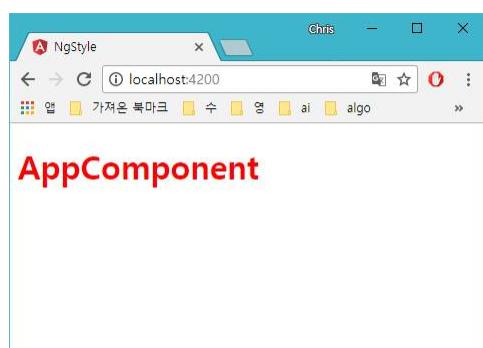
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  encapsulation: ViewEncapsulation.Emulated
})
export class AppComponent { }
```

### app.component.html

```
<h1>AppComponent</h1>
```

### app.component.css

```
h1 {
  color: red;
}
```



이 모드를 사용할 때 Angular는 `_ngcontent-*` 및 `_nghost-*` 라는 두 가지 고유 속성을 사용하여 각 구성 요소를 식별합니다. 이렇게 적용이 되면 쉐도우돔에 버금가게 CSS 설정을 컴포넌트별로 구분하여 적용할 수 있다.

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>NgStyle</title>
    <base href="/">

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <style type="text/css">
      /* You can add global styles to this file, and also import other style files */
    </style>
    <style>
      h1[_ngcontent-c0] {
        color: red;
      }
    </style>
  </head>
  <body>
    <app-root _nghost-c0="" ng-version="5.1.0">
      <h1 _ngcontent-c0="">AppComponent</h1>

      <router-outlet _ngcontent-c0=""></router-outlet>
    </app-root>
    <script type="text/javascript" src="inline.bundle.js"></script>
    <script type="text/javascript" src="polyfills.bundle.js"></script>
    <script type="text/javascript" src="styles.bundle.js"></script>
    <script type="text/javascript" src="vendor.bundle.js"></script>
    <script type="text/javascript" src="main.bundle.js"></script>
  </body>
</html>
```

## ViewEncapsulation.Native

이 캡슐화는 Angular가 특정 Component에 대해 Native Shadow DOM을 사용하도록 설정합니다. 브라우저에 따라 Shadow DOM v1이 적용됩니다. 네이티브 캡슐화를 활성화하기 전에 브라우저 지원을 고려하십시오.



<https://caniuse.com/#feat=shadowdomv1>

A screenshot of the 'Can I use...' website for the 'Shadow DOM v1' feature. The page header shows the URL <https://caniuse.com/#feat=shadowdomv1>. The main content area has a heading '# Shadow DOM v1' and a global usage statistic of '56.21% + 12.16% = 68.37%'. Below this is a detailed table of browser support:

IE	Edge	Firefox	Chrome	Safari	iOS Safari	Opera Mini	Chrome for Android	UC Browser for Android	Samsung Internet
			49	10.1	10.2				4
11	16	57	61	11	10.3				5
	17	58	62	11.2		all	62	11.4	6.2
		59	63	TP					
		60	65						

Below the table, there are links for 'Notes', 'Known issues (0)', 'Resources (8)', and 'Feedback'. A note at the bottom states: 'Shadow DOM v0 was implemented in Chrome/Opera but other browser vendors are implementing v1. MS Edge status: Under Consideration. Firefox status: in-development.' A warning note says: 'Certain CSS selectors do not work (:host > .local-child) and styling slotted content (:slotted) is buggy.'

## app.component.ts

```
import { Component, ViewEncapsulation } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  encapsulation: ViewEncapsulation.Native
})
export class AppComponent {

}
```

## 크롬 개발자모드(F12)에서 확인

```
...<!DOCTYPE html> == $0
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>NgStyle</title>
    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <style type="text/css">
      /* You can add global styles to this file, and also import other style files */
    </style>
  </head>
  <body>
    <app-root ng-version="5.1.0">
      <#shadow-root (open)>
        <style>h1 {
          color: red;
        }
        </style>
        <h1>AppComponent</h1>
        <router-outlet></router-outlet>
      </app-root>
      <script type="text/javascript" src="inline.bundle.js"></script>
      <script type="text/javascript" src="polyfills.bundle.js"></script>
      <script type="text/javascript" src="styles.bundle.js"></script>
      <script type="text/javascript" src="vendor.bundle.js"></script>
      <script type="text/javascript" src="main.bundle.js"></script>
    </body>
  </html>
```

## src/styles.css

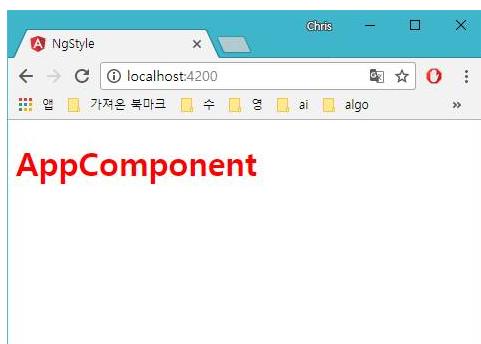
```
/* You can add global styles to this file, and also import other style files */
h1 {
  color: blue;
}
```

## src/index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>NgStyle</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
  <style media="screen">
    h1 {
      color: green;
    }
  </style>
</body>
</html>
```

index.html이나 styles.css에 설정된 CSS 설정은 쉐도우돔을 이용하는 컴포넌트 템플릿의 영향을 주지 않습니다.

브라우저가 지원하는 쉐도우돔 기술을 사용하면 컴포넌트별로 완벽한 디자인 적용의 분리를 구현할 수 있습니다.



## 크롬 개발자모드(F12)에서 확인

```
...<!DOCTYPE html> == $0
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>NgStyle</title>
    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
  <style type="text/css">
    /* You can add global styles to this file, and also import other style files */
    h1 {
      color: blue;
    }
  </style>
</head>
<body>
  <app-root ng-version="5.1.0">
    <#shadow-root (open)>
      <style>h1 {
        color: red;
      }
      </style>
      <h1>AppComponent</h1>
    </app-root>
    <style media="screen">
      h1 {
        color: green;
      }
    </style>
    <script type="text/javascript" src="inline.bundle.js"></script>
    <script type="text/javascript" src="polyfills.bundle.js"></script>
    <script type="text/javascript" src="styles.bundle.js"></script>
    <script type="text/javascript" src="vendor.bundle.js"></script>
    <script type="text/javascript" src="main.bundle.js"></script>
  </body>
</html>
```

h1 { color: green; } 설정은 설명을 위해 추가한 것 입니다.

## ViewEncapsulation.None

컴포넌트의 캡슐화를 모두 비활성화 할 수 있습니다. 이 모드를 사용하면 Angular는 정의 된 스타일을 <head> 태그에 추가하므로 이 캡슐화를 사용하여 스타일을 컴포넌트 간에 공유 할 수 있습니다.

### app.component.ts

```
import { Component, ViewEncapsulation } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent { }
```

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>NgStyle</title>
    <base href="/">

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <style type="text/css">
      /* You can add global styles to this file, and also import other style files */
      h1 {
        color: blue;
      }
    </style>
    <style>
      h1 {
        color: red;
      }
    </style>
  </head>
  <body>
    <app-root ng-version="5.1.0">
      <h1>AppComponent</h1>
    </app-root>
    <script type="text/javascript" src="inline.bundle.js"></script>
    <script type="text/javascript" src="polyfills.bundle.js"></script>
    <script type="text/javascript" src="styles.bundle.js"></script>
    <script type="text/javascript" src="vendor.bundle.js"></script>
    <script type="text/javascript" src="main.bundle.js"></script>
  </body>
</html>
```

app.component.css 설정이 head 태그 하단으로 추가 되었습니다. 추가 선택자가 없는 상태이므로 index.html이나 styles.css 파일에 설정한 CSS 와 동일한 효력을 갖게 됩니다.