



시작 합니다~



정의

Angular

What is Angular?



TypeScript 기반

오픈 소스 프론트엔드

웹 애플리케이션 프레임워크

또는 웹 플랫폼



특징

What is Angular?



1. 개선된 개발 생산성

- **컴포넌트 기반 개발** - *AngularJS*는 *Controller*와 *\$scope*가 개발의 중심이었지만 *Angular*에서는 컴포넌트가 개발의 중심이다. 컴포넌트 기반 개발(CBD: *Component Based Development*)은 개발 생산성을 높이며 대규모 애플리케이션에 적합한 구조이다.
- **TypeScript의 도입**
- **개발 도구의 통합 및 개발 환경 구축 자동화** - *Angular*는 *Angular CLI*를 통해 간편한 개발 환경 구축을 지원한다. 간단한 명령어를 사용하여 프로젝트 생성에서 빌드, 테스트, 구성요소 추가 등을 간편하게 할 수 있으며, *개발용 서버*를 내장하고 있어 실행까지 할 수 있다. *Angular CLI*는 개발환경 구축에 소요되는 시간을 최소화할 수 있어서 개발작업에 집중할 수 있도록 돕는다.

Angular

What is Angular?



2. 성능의 향상

- *Digest Loop*로 인한 성능저하 문제의 해결
- *AoT 컴파일*
- *Lazy Loading*
- *코드 최적화*



역사



History

`2016년

앵귤러(Angular, Angular 2+ 또는 Angular v2 이상)

앵귤러는 마이크로소프트의 타입스크립트 언어의 사용을 권고하며 다음의 기능을 도입함.

- 클래스 기반 객체 지향 프로그래밍
- 정적 타이핑
- 제네릭

·
·
·

`2010년

AngularJS, 싱글 페이지 애플리케이션 개발 중에 마주치는 여러 문제들을 해결하기 위해 개발



History

- *AngularJS*의 Controller와 *\$scope* 기반 개발에서 컴포넌트 기반 개발(**CBD**, Component Based Development)로 전환되었다.
- *AngularJS*의 *angular.module*과 *jQuery*보다 향상된 모듈 시스템과 DOM 제어 기능을 제공하며 API 또한 단순화되었다.
- 선택적 **데이터 바인딩**(one-way, two-way...)을 지원하고 **디렉티브**(directive)와 **서비스, 의존성 주입**(dependency injection)은 간소화 되었다.
- 주력 개발 언어로서 **TypeScript**를 도입하여 대규모 개발에 적합한 **정적 타입과 인터페이스, 제네릭** 등 **타입 체크 지원** 기능을 제공한다.
- **ECMAScript6**에서 새롭게 도입된 모듈, 클래스 등과 **ECMAScript7**의 데코레이터를 지원한다.
- 강력한 개발환경 지원 도구인 **Angular CLI**를 제공한다.

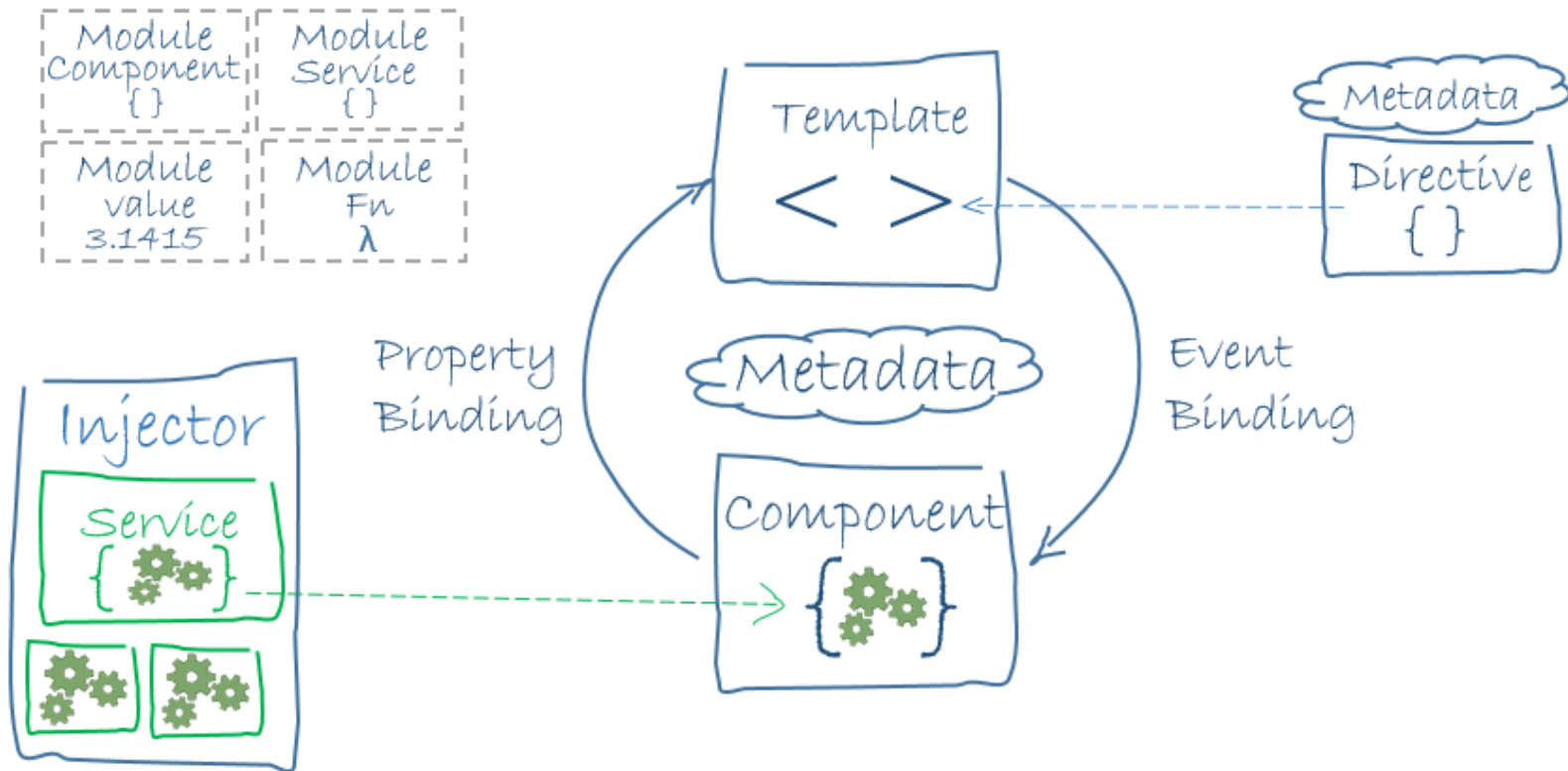


아키텍처

Angular



Architecture



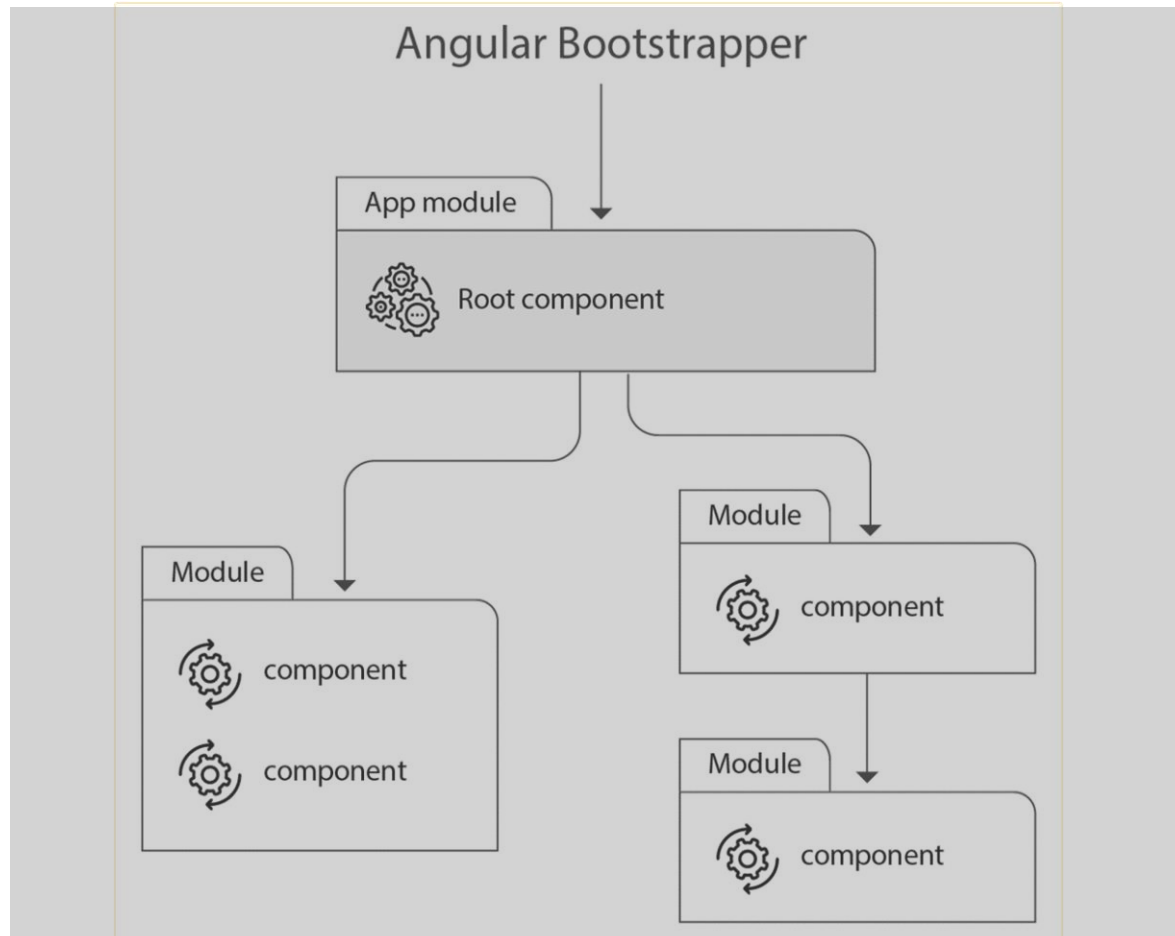
“주요 빌딩 블록” - 모듈, 컴포넌트, 템플릿, 메타데이터,
데이터 바인딩, 디렉티브, 서비스, 의존성 인젝션



모듈

Angular

Module

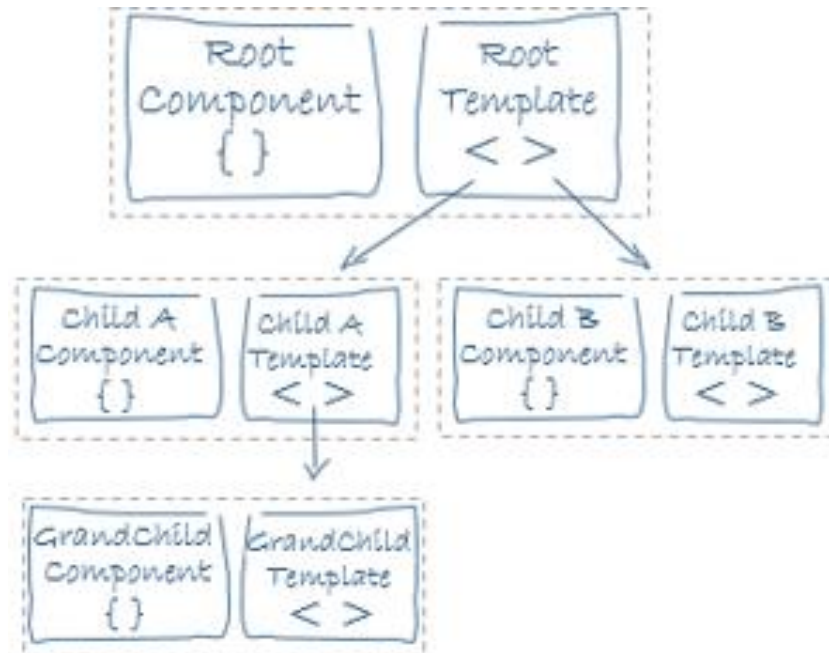




컴포넌트



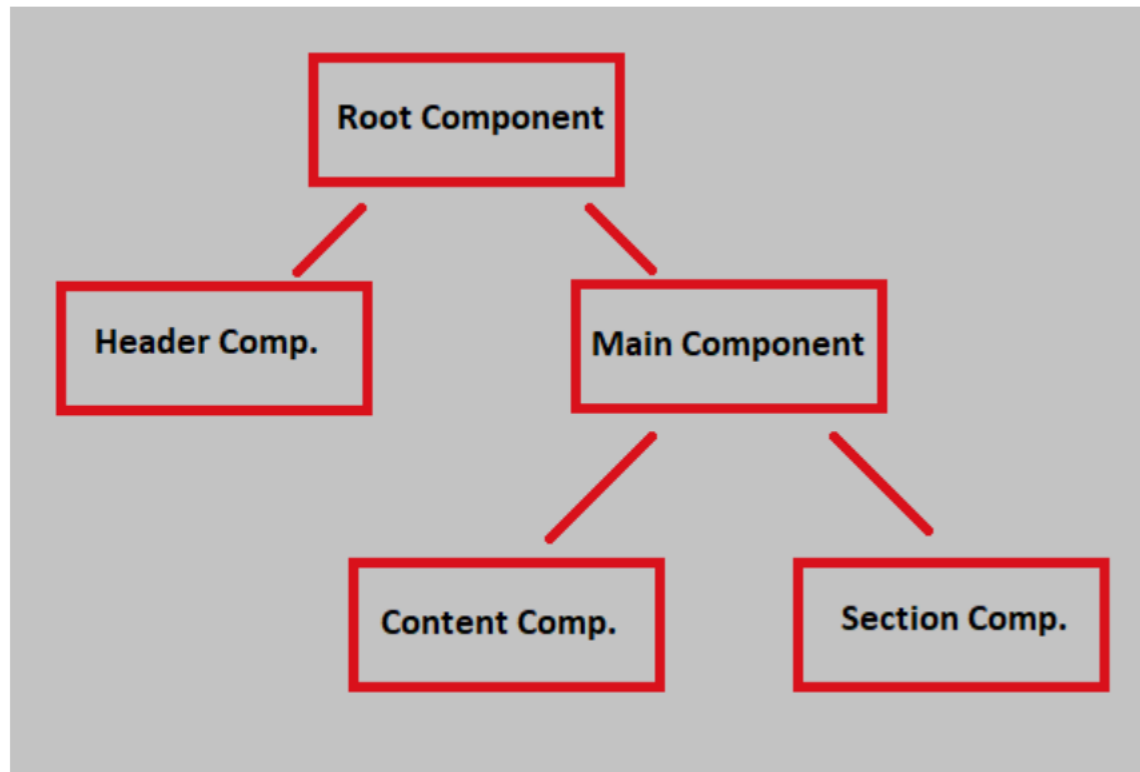
Component Hierarchy



컴포넌트 - 부모, 자식 계층 구조

Angular

Component Hierarchy



Example of Angular Tree Structure

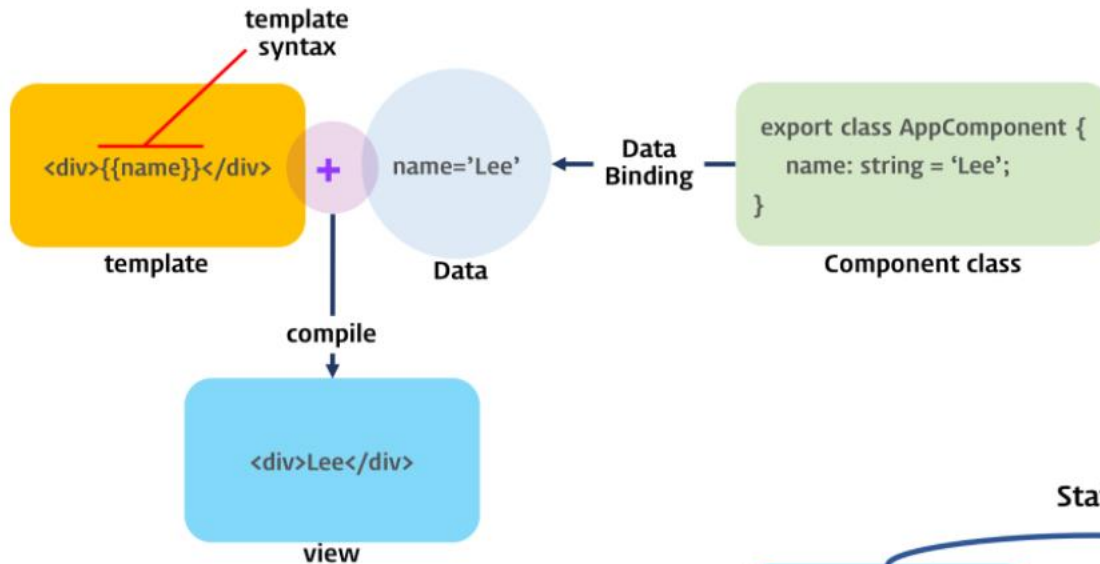


템플릿

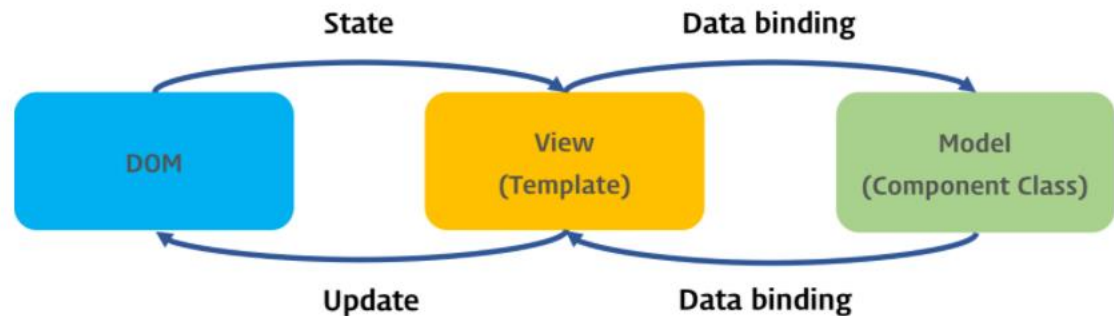
Angular



Template



템플릿의 뷰 생성 과정



Angular의 뷰와 모델

Angular



Template

index.html

```
<!doctype html>
<html lang="en">
<body>
  <game-root></game-root>
</body>
</html>
```

app.component.ts

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

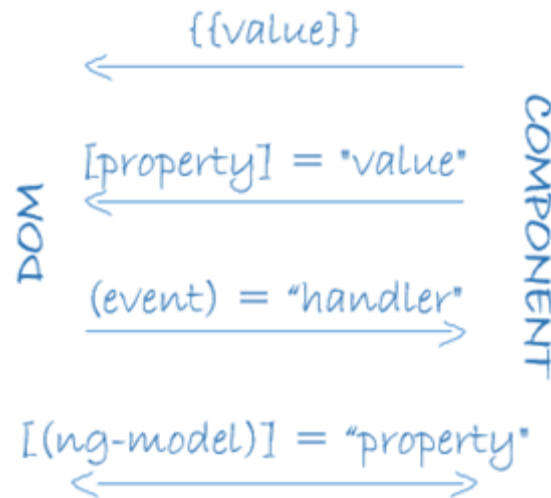
@Component({
  selector: 'game-root',
  template: `
    <div><h1>{{pageTitle}}</h1>
    <div>Game app root component</div>
  </div>`
})
export class AppComponent {
  pageTitle = 'Angular Games Viewer';
}
```



데이터 바인딩

Angular

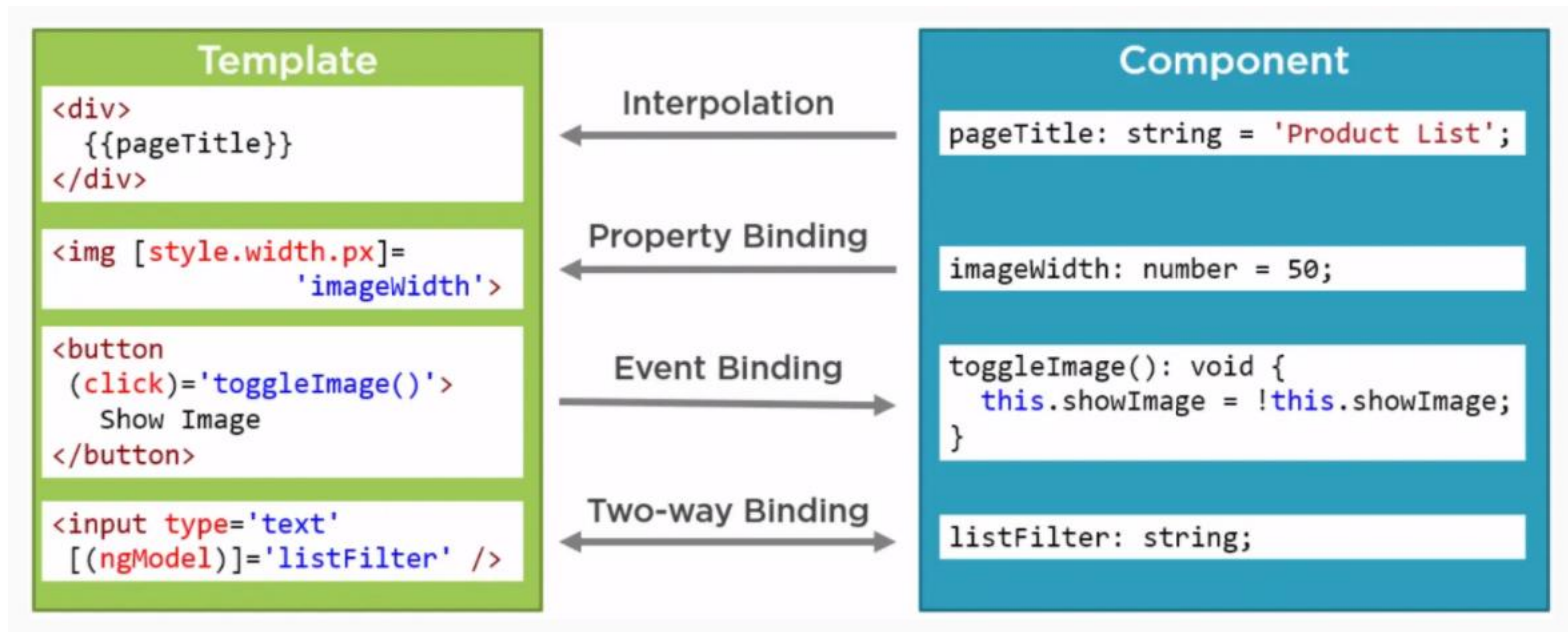
Data Binding



Angular는 **템플릿**과 **컴포넌트**를 일치시키는 **데이터 바인딩**을 지원 - HTML템플릿에 **바인딩 마크업**을 추가하는것으로 Angular에게 어떻게 둘을 연결할 지를 알려줄수 있다.

Angular

Data Binding



Angular

Data Binding



```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-hello',
5    template: `
6      <h2>안녕하세요 {{name}}</h2>
7      <input type="text" placeholder="이름을 입력하세요" #inputYourName
8      <button (click)="setName(inputYourName.value)">등록</button>
9    `,
10   styles: [ ... ],
11   })
12
13   export class HelloComponent {
14     name: string;
15
16     setName(name: string) {
17       this.name = name;
18     }
19   }
```

컴포넌트 클래스와 템플릿의 연동

Data Binding



데이터 바인딩	데이터의 흐름	문법
인터플레이션	컴포넌트 클래스 \Rightarrow 템플릿	<code>{{ expression }}</code>
프로퍼티 바인딩	컴포넌트 클래스 \Rightarrow 템플릿	<code>[property]="expression"</code>
어트리뷰트 바인딩	컴포넌트 클래스 \Rightarrow 템플릿	<code>[attr.attribute-name]="expression"</code>
클래스 바인딩	컴포넌트 클래스 \Rightarrow 템플릿	<code>[class.class-name]="expression"</code>
스타일 바인딩	컴포넌트 클래스 \Rightarrow 템플릿	<code>[style.style-name]="expression"</code>
이벤트 바인딩	컴포넌트 클래스 \leftarrow 템플릿	<code>(event)="statement"</code>
양방향 데이터 바인딩	컴포넌트 클래스 \leftrightarrow 템플릿	<code>[(ngModel)]="property"</code>

Angular




Data Binding

```
1  $(function() {  
2    var title = 'app works!';  
3    $('h1').text(title);  
4  });
```

javascript

DOM



jQuery에 의한 DOM 조작(Procedural programming)

```
8  export class AppComponent {  
9    title = 'app works!';  
10 }
```

컴포넌트 클래스

템플릿



데이터 바인딩에 의한 템플릿과 컴포넌트 클래스의 연결(Declarative programming)

<Interpolation>

Data Binding



```
@Component({
  selector: 'app-root',
  template: `
    <!-- input 요소의 value 프로퍼티에 컴포넌트 클래스의 name 프로퍼티 값을 프로퍼티 바인딩 -->
    <input type="text" [value]="name">

    <!-- p 요소의 innerHTML 프로퍼티에 컴포넌트 클래스의 contents 프로퍼티 값을 프로퍼티 바인딩 -->
    <p [innerHTML]="contents"></p>

    <!-- img 요소의 src 프로퍼티에 컴포넌트 클래스의 imageUrl 프로퍼티 값을 프로퍼티 바인딩 -->
    <img [src]="imageUrl"><br>

    <!-- button 요소의 disabled 프로퍼티에 컴포넌트 클래스의 isDisabled 프로퍼티 값을 프로퍼티 바인딩 -->
    <button [disabled]="isDisabled">disabled button</button>
  `
})
```

<Property Binding>

Angular

Data Binding



```
@Component({
  selector: 'app-root',
  template: `
    <!-- 프로퍼티 바인딩 -->
    <input id="user" type="text" [value]="name">
    <!-- 어트리뷰트 바인딩 -->
    <input id="user" type="text" [attr.value]="name">
  `
})
```

<Attribute Binding>

Angular

Data Binding



```
@Component({
  selector: 'app-root',
  template: `
    <!-- (1) -->
    <input type="text" [value]="name" (input)="setName($event)">
    <!-- (2) -->
    <button (click)="clearName()">clear</button>
    <!-- (3) -->
    <p>name: {{ name }}</p>
  `
})
```

<Event Binding>

Angular

Data Binding



```
<element [class.class-name]="booleanExpression"> ... </element>  
<element [class]="class-name-list"> ... </element>
```

<Class Binding>

```
<element [(ngModel)]="property"> ... </element>
```

<Two-way Data Binding>



파이프

Angular



Pipe (default)

common

P AsyncPipe

P DecimalPipe

P JsonPipe

P PercentPipe

P UpperCasePipe

P CurrencyPipe

P I18nPluralPipe

P KeyValuePipe

P SlicePipe

P DatePipe

P I18nSelectPipe

P LowerCasePipe

P TitleCasePipe

```
@Component({
  selector: 'json-pipe',
  template: `<div>
    <p>Without JSON pipe:</p>
    <pre>{{object}}</pre>
    <p>With JSON pipe:</p>
    <pre>{{object | json}}</pre>
  </div>`
})
export class JsonPipeComponent {
  object: Object = {foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}};
}
```

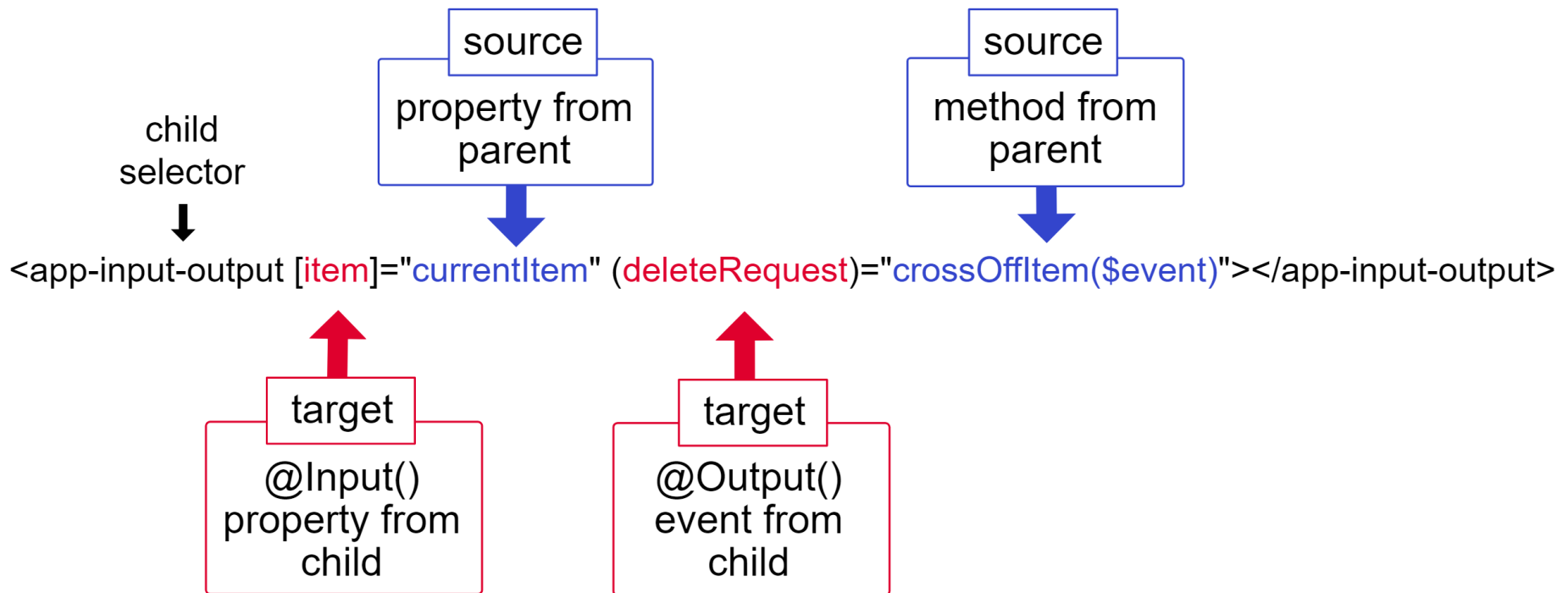


공유

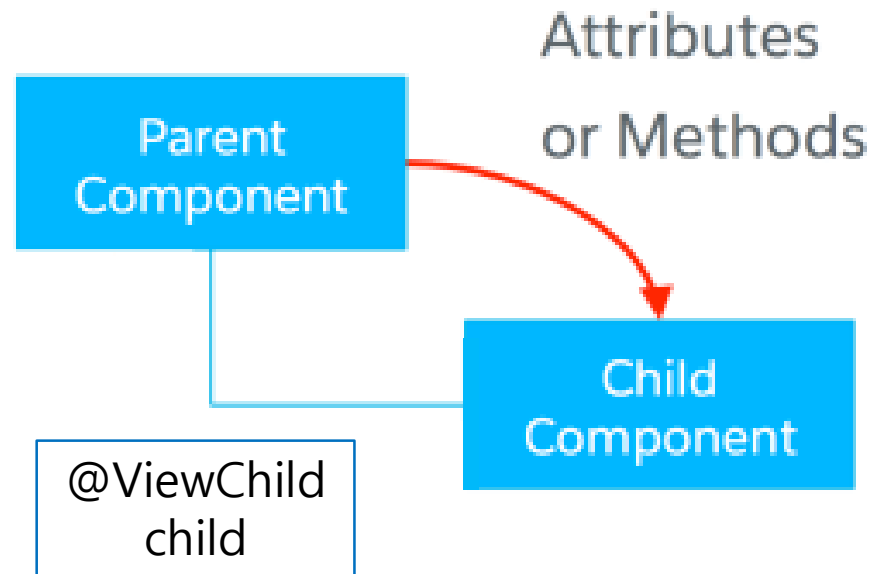
Angular



Sharing data between child and parent



Accessing a child and call methods or access variable



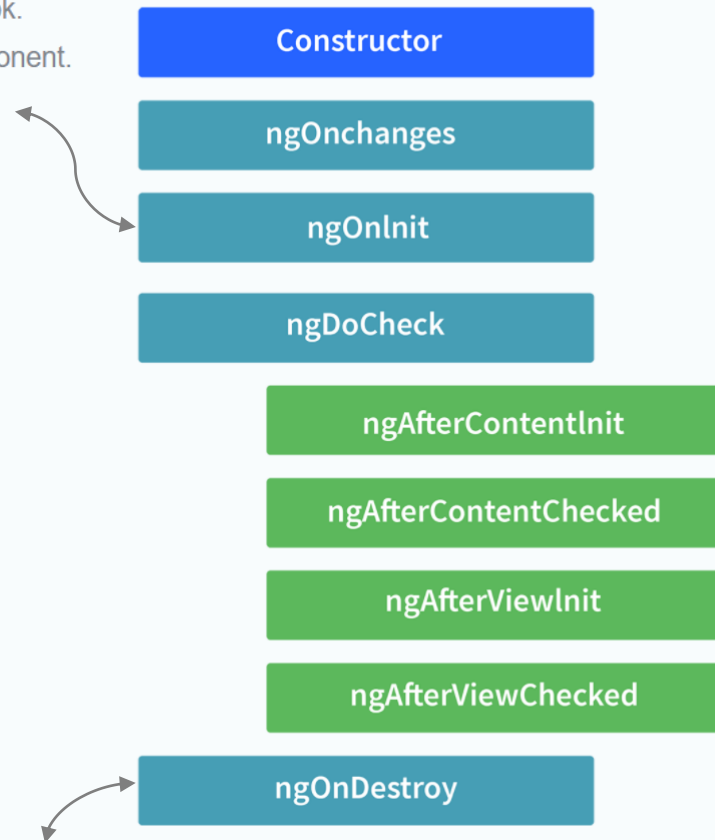


라이프사이클



ngOnInit() This hook gets called once, after the **ngOnChanges** hook. It initializes the component and sets the input properties of the component.

Component Lifecycle Hooks



ngOnDestroy() It gets called just before Angular destroys the component. This hook can be used to clean up the code and detach event handlers.



네이밍 규칙

Angular



네이밍 컨벤션

기능을 명확히 설명하는 구성요소의 이름.구성요소 타입.ts

`/src/app/{{업무}}/component/{{업무모듈명}}.component.ts`

`/src/app/{{업무}}/component/{{업무모듈명}}.component.html`

`/src/app/{{업무}}/service/{{업무모듈명}}.service.ts`

`/src/app/{{업무}}/{{업무모듈명}}.module.ts`

...



설치해 봅시다!

Angular



Install

- **Node.js & NPM** : <https://nodejs.org/en/>
 - **TypeScript** : *\$ npm install -g typescript*
 - **Angular CLI** : *\$ npm install -g @angular/cli*
 - **Visual Studio Code** : <https://code.visualstudio.com/download>
-

Angular



Install

version check

```
$ node -v  
> v12.18.3
```

```
$ npm -v  
> 6.14.6
```

```
$ yarn -v  
> 1.22.5
```

```
$ tsc -v  
> Version 2.8.3
```

```
$ ng version  
> Angular CLI: 10.1.2
```



Command

Angular



Command

프로젝트 생성

```
$ ng new {project}
```

Working Directory 이동

```
$ cd {project}
```

VSCode 오픈

```
$ code .
```

Development Server 구동

```
$ ng serve --open --port={port} --proxy-config proxy.conf.js
```

Component 생성

```
$ ng generate component {component}
```

Service 생성

```
$ ng generate service {service}
```

Route 생성

```
$ ng generate module app-routing --flat --module=app
```

빌드

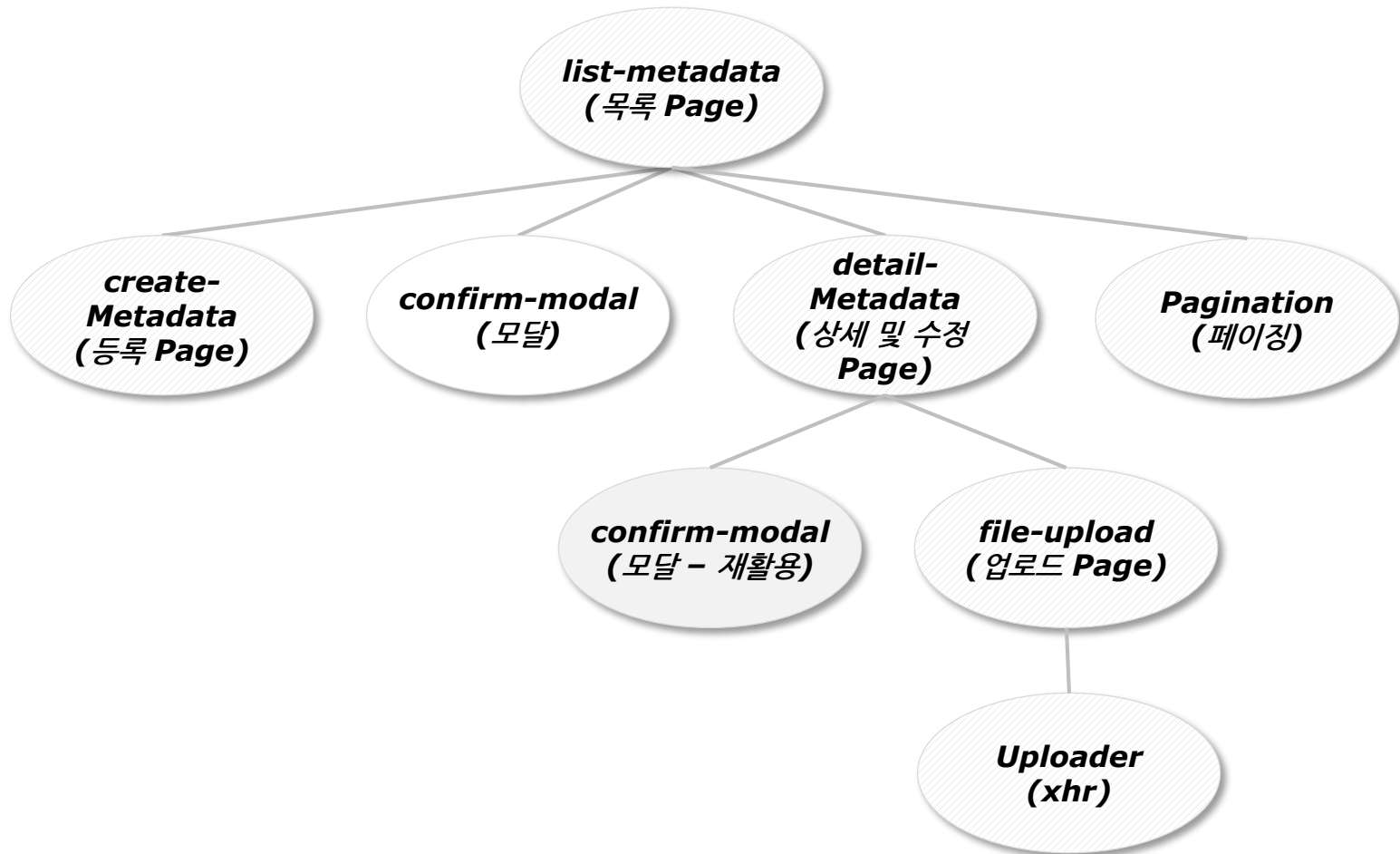
```
$ ng build --prod
```



Sample



예제 - Tree 컴포넌트 구조





TO DO...

Angular

기타 - 다른 주제들...



Router

***Sharing data
between
Components***

test

***Angular
material***

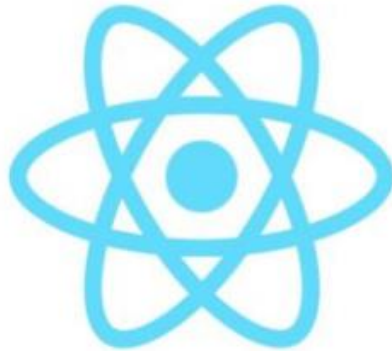
RxJS

e2e

The Three Kingdoms of Vue React Angular



VS



VS





:wq!