Angular

Petite histoire

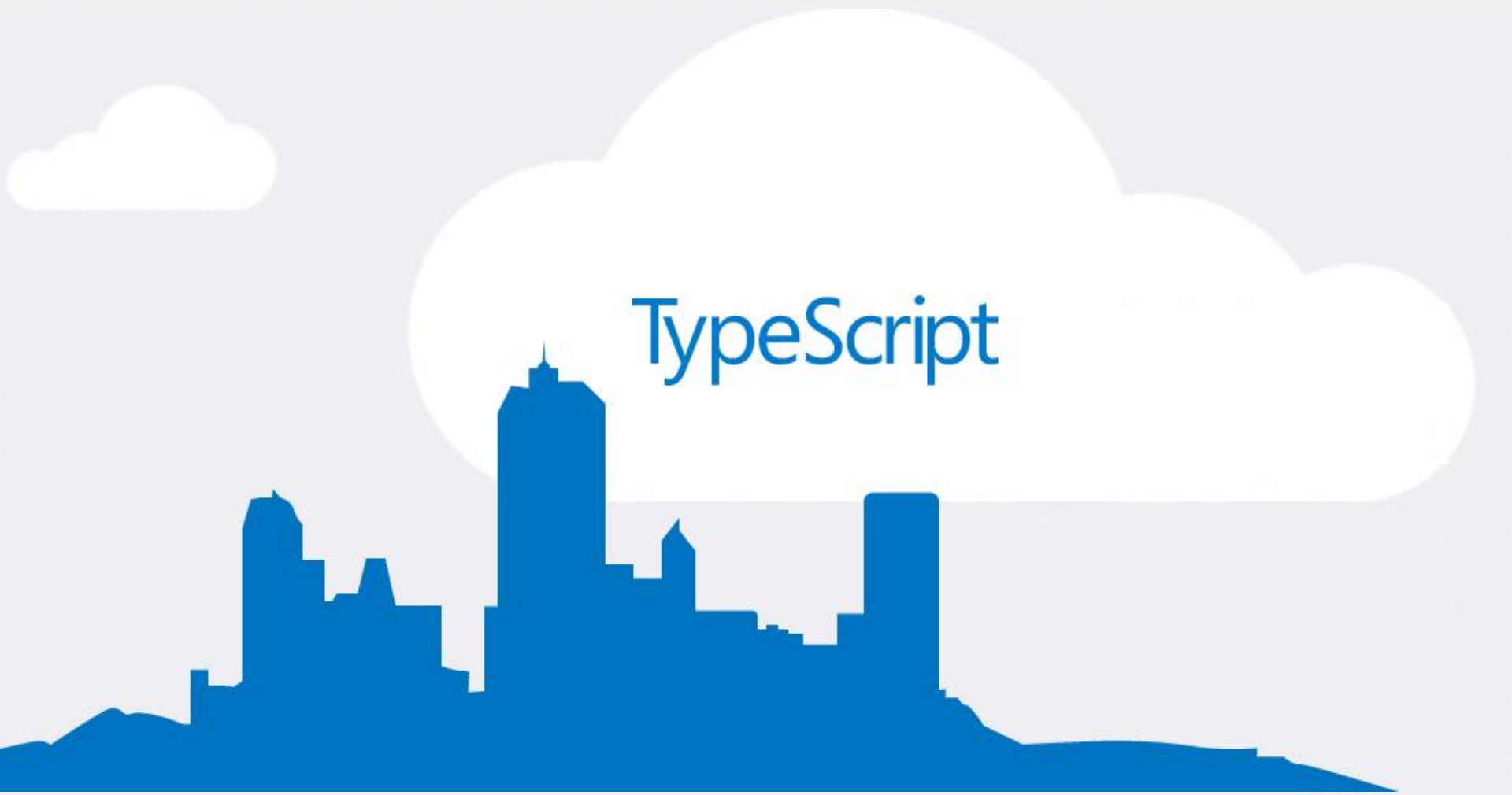
- 2009 - Création de la version originale par Misko Hevery
- 2010 - Misko rejoint Google qui soutient officiellement AngularJS
- 2013 - Explosion de la notoriété
- 2016 - Sortie d'Angular 2
- Framework Javascript pour faire des SPA
- Plusieurs langages : ES5, ES6, TypeScript et Dart
- Contient routeur, requêteur HTTP, gestion des formulaires, i18n, …
- Modulaire : découpé en sous paquets, nos apps sont découpées en composants et modules
- 5 fois plus rapide que Angular 1
- Tout est composant

Présentation

- La vocation d’Angular 2 : une plateforme pour le développement d’applications web et mobiles
- Le noyau de la librairie est modulaire. (On n’installe que ce dont on a besoin.)
- L’outillage a été amélioré, avec des outils comme TypeScript, Angular CLI, Augura...
- Angular s’exécute facilement dans plusieurs environnements :
  - sur le serveur avec Angular Universal
  - sur mobile avec Angular Mobile Toolkit ou NativeScript, etc...
- https://universal.angular.io/
- https://mobile.angular.io/

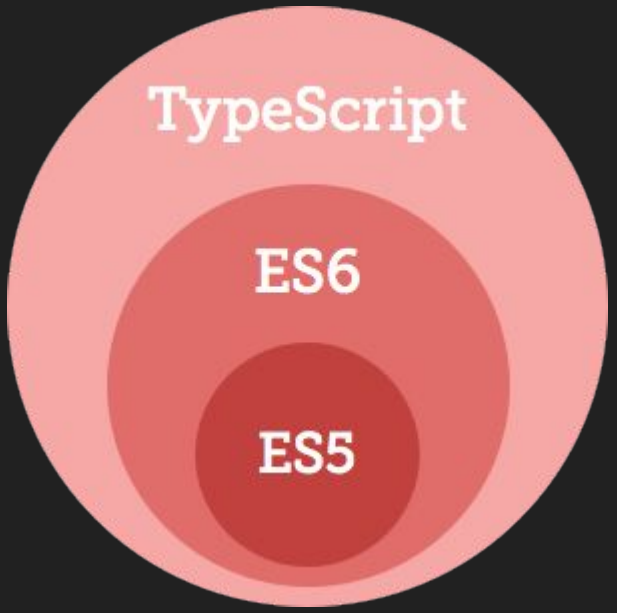
JavaScript

- Langage de script orienté prototype
- Interprété par le navigateur et quelques runtimes (NodeJS)
- Histoire :
  - 1995 – Sun/Netscape annonce JavaScitpt (ancien LiveScript)
  - 1996 – JavaScript dans Netscape 2.0 / JScript dans IE3
  - 1997 – Standard ECMAScript
  - 1998 – Adobe : ActionScript
- ECMA : Organisme privé européen de standardisation


TypeScript

TypeScript

- Créé par Microsoft en 2012, open-source, qui transpile vers JavaScript.
- Sur-ensemble d'ES6 (aka ES2015). Tout JavaScript est donc du TypeScript valide.
- Principales caractéristiques :
  - types, interfaces, classes, décorateurs, modules, fonctions fléchées, templates chaîne.
- Supporté par de nombreuses librairies JavaScript tierce-partie.
- Supporté par plusieurs IDE : WebStorm/IntelliJ Idea, Visual Studio Code, Sublime Text, etc...



Variables TypeScript

```
var isDone: boolean = false;
var height: number = 6;
var name: string = "dave";
var myList number[] = [1, 2, 3]; // option #1
var myList:Array<number> = [1, 2, 3]; // option #2
var changeMe: any = 4;
changeMe = "I'm a string now";
var myList any[] = [3, true, "pizza"];
```

Fonctions TypeScript

- void return type:

```
function myLogger(msg?:string): void {
   console.log('My custom logger: ' + msg);
}
```

- Generics:

```
function identity<T>(arg: T): T {
   return arg;
}
// output has 'string' type (explicit/inferred):
var output = identity<string>("Dave");
var output = identity("Dave");
```

- any return type : Désactive le typage

```
function selectSomething(x): any {
    if (typeof x == "object") {
        return 10;
    } else {
        return "abc";
    }
```

Interfaces et classes TypeScript

```
interface UserIface {
  name: string;
  weight?: number; // optional
}

function displayUser(user: UserIface) {
  console.log(user.name);
}
```

ça marche même sans “weight” dans l’interface

```
var aUser = {name: "John Smith", weight 200};
displayUser(aUser);
```

Classes TypeScript

```
class User extends UserIface {
    fname: string;
    lname: string;

    constructor(fname:string, lname:string) {
        this.fname = fname;
        this.lname = lname;
    }

    fullname():string {
       return this.fname + " " + this.lname;
    }
}
```

```
var u1:User, u2:User;
u1 = new User("Jane", "Smith");
u2 = new User("John", "Jones");
console.log("user1 = " + u1.fullname());
console.log("user2 = " + u2.fullname());
```

Seulement un seul constructeur par classe
On peut implémenter plusieurs interfaces

Angular 2 et TypeScript

- Symbole `@` pour les annotations/decorators
- `@Component ({selector, template, ... })`
- Une classe typique est AppComponent dans app.component.ts
- Un module dans app.module.ts
- Bootstrap du composant racine dans main.ts

Les concepts

- Directives
  - Encapsulation de logique basique IHM.
  - $ ng g directive MaDirective
- Components
  - Une directive avec une vue
  - $ ng g component MonComposant
- Pipes
  - Transformation de données
  - $ ng g pipe monFiltre
- Services
  - Simple classe pouvant être injectée
  - $ ng g service monDAO
- Dependency Injection

Directive

```
@Directive({selector: '[tooltip]', inputs: ['tooltip'], host: { '(mouseenter)': 'onMouseEnter()', '(mouseleave)': 'onMouseLeave()' }})
export class Tooltip {
    private overlay:Overlay;
    private tooltip:string;
    constructor(private el:ElementRef, manager: OverlayManager) {
        this.el = el;
        this.overlay = manager.get();
    }
    onMouseEnter() {
        this.overlay.open(this.el, this.tooltip);
    }
    onMouseLeave() {
        this.overlay.close();
    }
}
```

Directive

```
@Directive({selector: '[tooltip]', inputs: ['tooltip'], host: { '(mouseenter)': 'onMouseEnter()', '(mouseleave)': 'onMouseLeave()' }})
export class Tooltip {
    private overlay:Overlay;
    private tooltip string;
    constructor(private el ElementRef, manager: OverlayManager) {
        this.el = el;
        this.overlay = manager.get();
    }
    onMouseEnter() {
        this.overlay.open(this.el, this.tooltip);
    }
    onMouseLeave() {
        this.overlay.close();
    }
}
```

Component

```
@Component({selector: 'hello-world'})

@View({
    template: '<h1>Hello, {{this.target}}!</h1>'
})

class HelloWorld {
    target string;
    constructor() {
        this.target = 'world';
    }
}
```

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

@Component({
  selector: 'app-hello-world',
  templateUrl: './hello-world.component.html',
  styleUrls: ['./hello-world.component.css']
})
export class HelloWorldComponent implements OnInit {

  constructor() { }

  ngOnInit() {}
}
```

Component

```
@Component({selector: 'hello-world'})

@View({
    template: '<h1>Hello, {{this.target}}!</h1>'
})

class HelloWorld {
    target:string;
    constructor() {
        this.target = 'world';
    }
}
```

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

@Component({
  selector: 'app-hello-world',
  templateUrl: './hello-world.component.html',
  styleUrls: ['./hello-world.component.css']
})
export class HelloWorldComponent implements OnInit {

  constructor() { }

  ngOnInit() {}
}
```

Pipe

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

@Pipe({ name: 'lowercase' })
class LowerCasePipe implements PipeTransform {

    transform(value  string): string {
        if (!value) return value;
        if (typeof value !== 'string') {
            throw new Error('Invalid pipe value', value);
        }
        return value.toLowerCase();
    }
}
```

Service et injection de dépendance

```
import {Injectable} from "@angular/core";
import {Response} from "@angular/http";
import {Router} from "@angular/router";
import {ToasterService} from "angular2-toaster";
@Injectable()
export class ApiService {
    protected toasterService: ToasterService;

    constructor(toasterService: ToasterService) {
        this.toasterService = toasterService;
    }
    extractData(res: Response): any {
        return res.json();
    }
}
```

Angular 2 et TypeScript

- Utilisation d'Angular CLI pour générer un squelette d'application :
  - `[sudo] npm install -g angular-cli`
- Créer une application angular 2
  - `ng new myapp`
  - `cd myapp`
  - `ng serve`
- Extension Chrome DevTools : Angular Augry : https://augury.angular.io/
- Package Manager : npm
- Module Loader : SystemJS
- Transpiler : Traceur / TypeScript
- Build Tool : Broccoli

Demo Helloworld

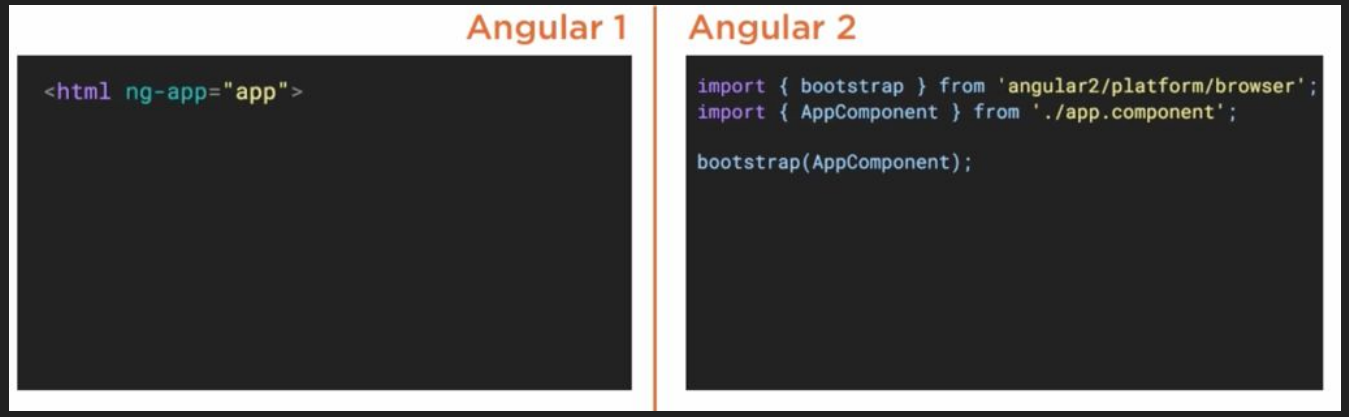
$ ng new helloWorld

Angular 2 vs Angular 1

Angular 1	Angular 2
Framework	Plateforme
JavaScript	TypeScript
Pattern Modèle-Vue-*	Pattern Composant
Liaison de données principalement BIdirectionnelle	Liaison de données principalement UNIdirectionnelle
Scope	Bye bye le scope
Injection de dépendance : plusieurs syntaxes possibles	Injection de dépendance : syntaxe unique.
API complexe	API simplifiée
Rendering normal	Rendering 5 fois plus rapide
Plusieurs “bonnes pratiques” concurrentes par la communauté	Bonnes pratiques officielles : https://angular.io/styleguide

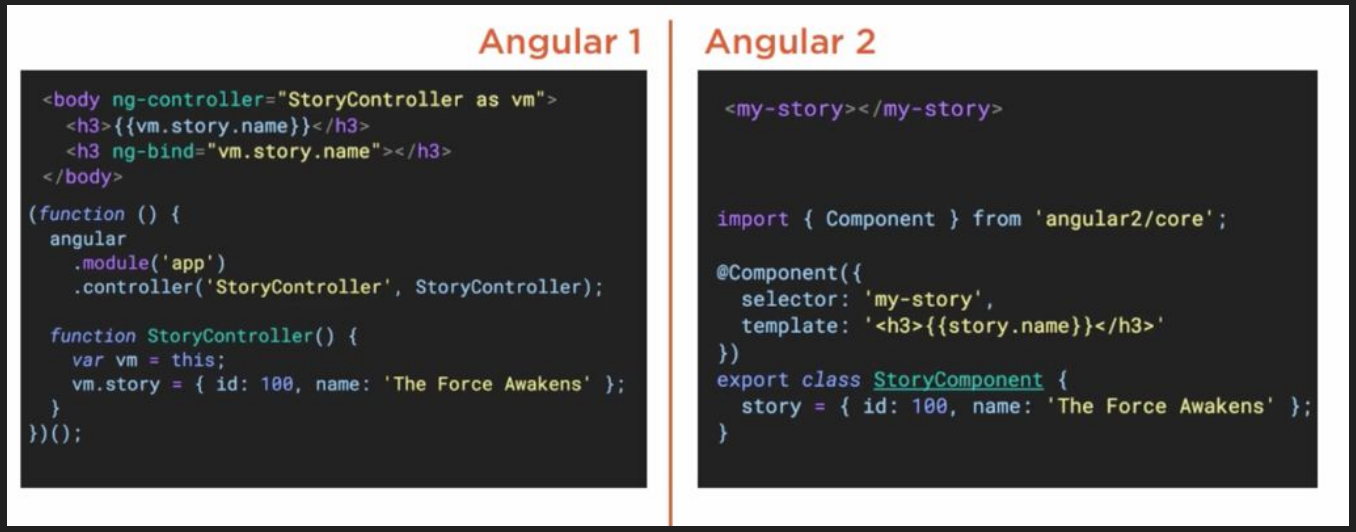
1. Bootstraper Angular

- NG1: directive ng-app (bootstrap automatique).
- NG2: bootstrap via code avec la fonction bootstrap()



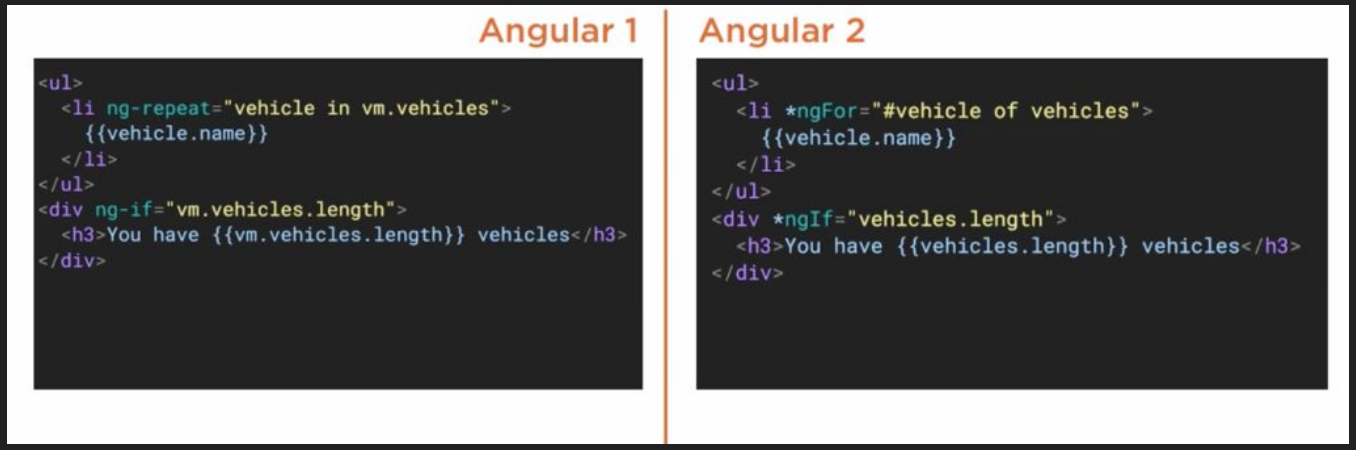
2. Des contrôleurs aux composants

- NG1: angular.controller()
- NG2: Classe avec décorateur @Component



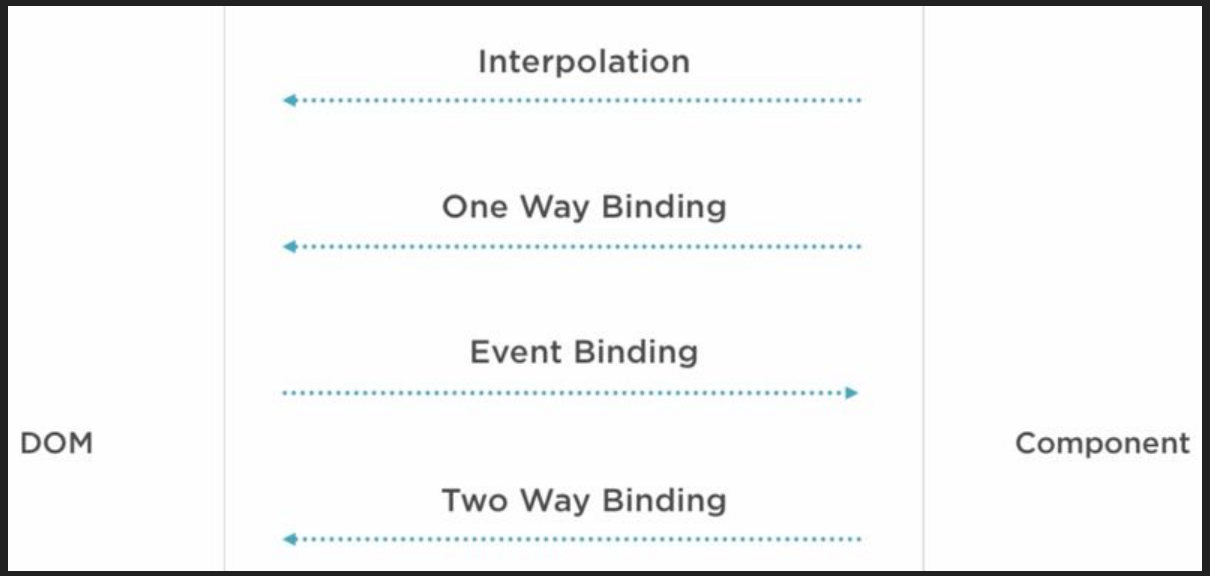
3. Directives structurelles

- NG1: Beaucoup de directives structurelles. Ici, ng-repeat et ng-if.
- NG2: Seules quelques directives conservées (comme *ngFor et *ngIf). Points importants : notation camelcase, étoile * devant nom de la directive (signale une directive structurelle), syntaxe let vehicle of vehicules (of et non pas in).



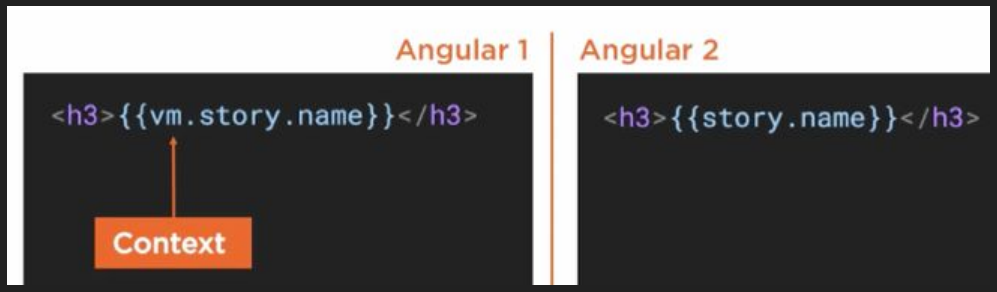
4. Liaison de données

- Permet de synchroniser les données entre les composants et le DOM (aka la vue).



4. Liaison de données

- Interpolation



- Binding de propriété (unidirectionnel)

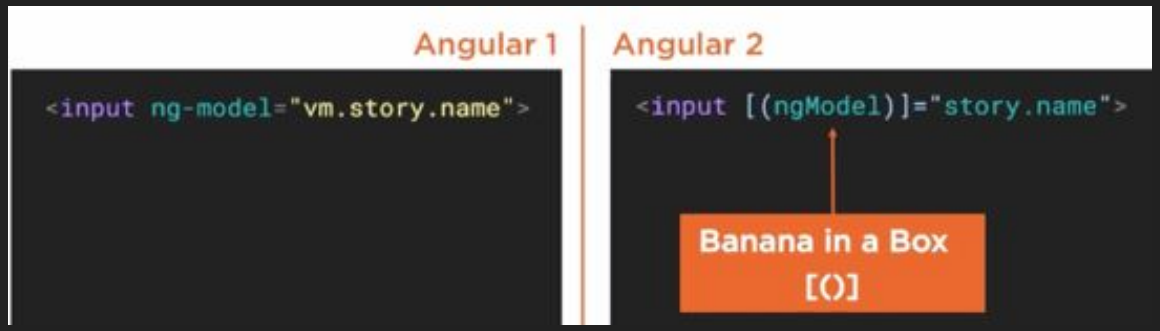
4. Liaison de données

- Binding d’événement
  - NG1: ng-click, ng-blur... — Directives custom Angular.
  - NG2: (click), (blur) — Fini les directives custom, on utilise les événements natifs d'un HTMLElement entre parenthèses.

Angular 1	Angular 2
ng-click="saveVehicle(vehicle)"	(click)="saveVehicle(vehicle)"
ng-focus="log('focus')"	(focus)="log('focus')"
ng-blur="log('blur')"	(blur)="log('blur')"
ng-keyup="checkValue()"	(keyup)="checkValue()"

4. Liaison de données

- Liaison de données bidirectionnelle (champ de formulaire)



5. Moins de directives

- NG1: ng-style, ng-src, ng-href...
- NG2: Plus de 40 directives NG1 ont disparu dans NG2 !

Angular 1

```
<div ng-style=
  "vm.story ?
    {visibility: 'visible'}
    : {visibility: 'hidden'}">

  <img ng-src="{{vm.imagePath}}">
  <br/>
  <a ng-href="{{vm.link}}">
    {{vm.story}}
  </a>

</div>
```

Angular 2

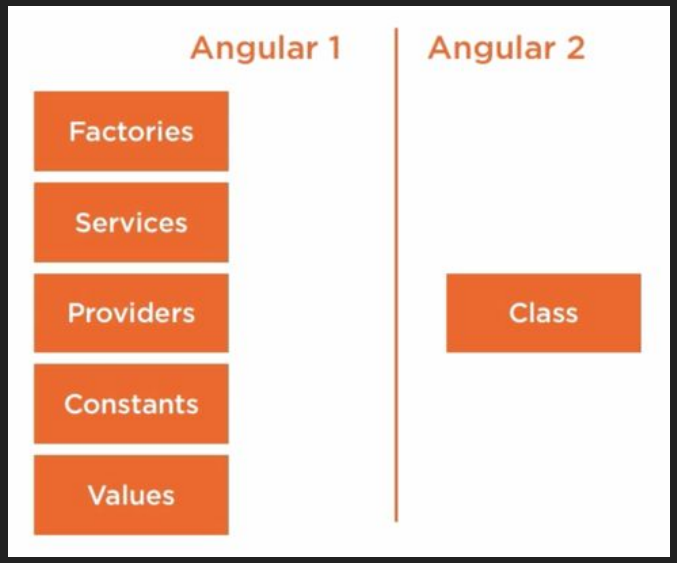
```
<div [style.visibility]=
  "story ? 'visible' : 'hidden'">

  <img [src]="imagePath">
  <br/>
  <a [href]="link">{{story}}</a>

</div>
```

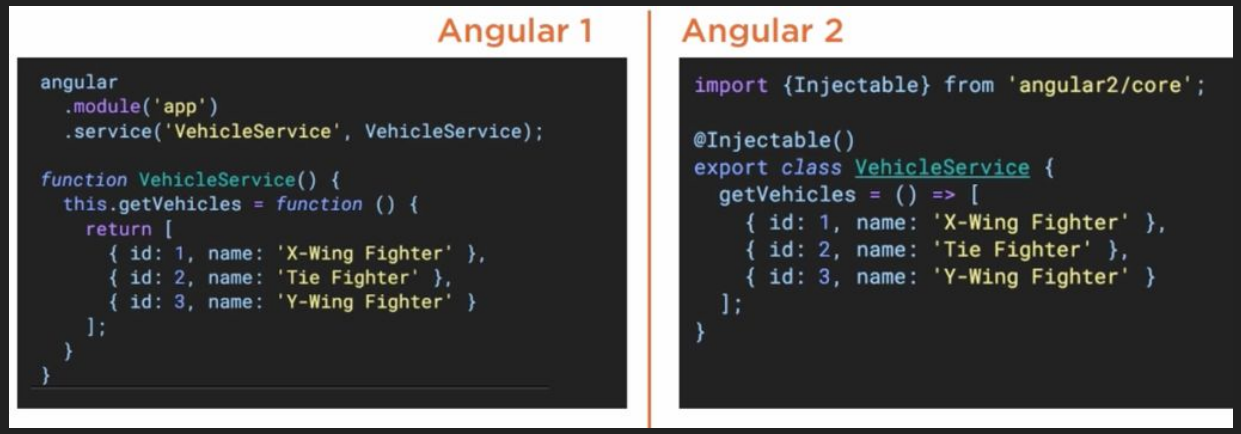
6. Services et DI (1/2)

- NG1: Les données ou fonctionnalités partagées utilisent des factories, des services, des providers…
- NG2: Un seul concept subsiste : une classe TypeScript.



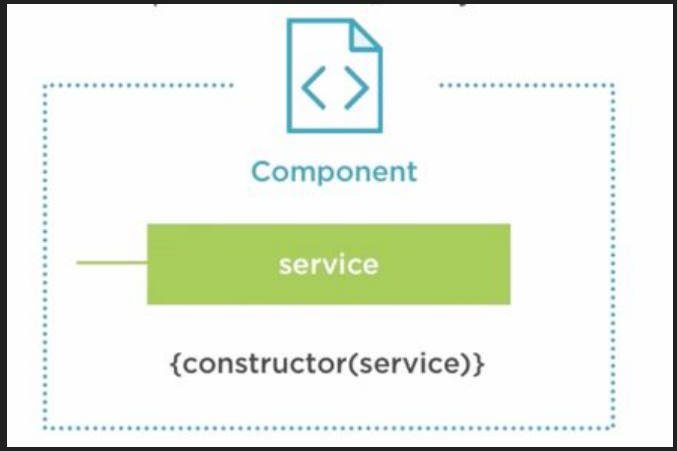
6. Services et DI (2/2)

- NG1: Ici, angular.service(), mais pourrait être angular.factory(), angular.provider()...
- NG2: Simple classe avec le décorateur @Injectable.



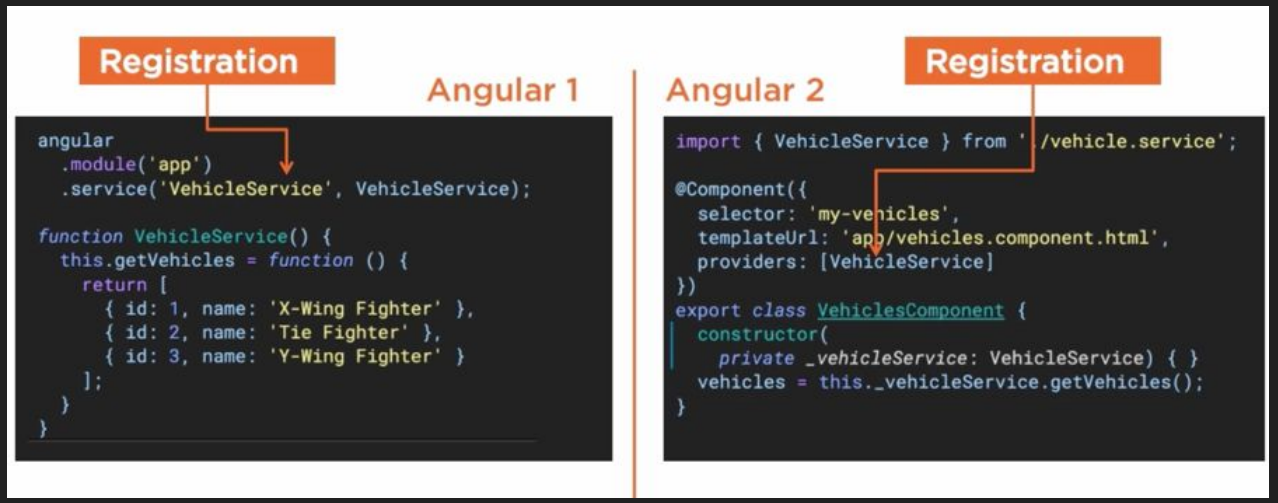
7. Injection de dépendance

- Dans Angular, les services contiennent toute la logique applicative. Exemple : service qui récupère les données du serveur via un appel HTTP.
- Lorsqu'un composant a besoin d'utiliser un service, il utilise l'injection de dépendance (DI).
- La DI se fait en deux temps :
  - Déclaration
  - Injection



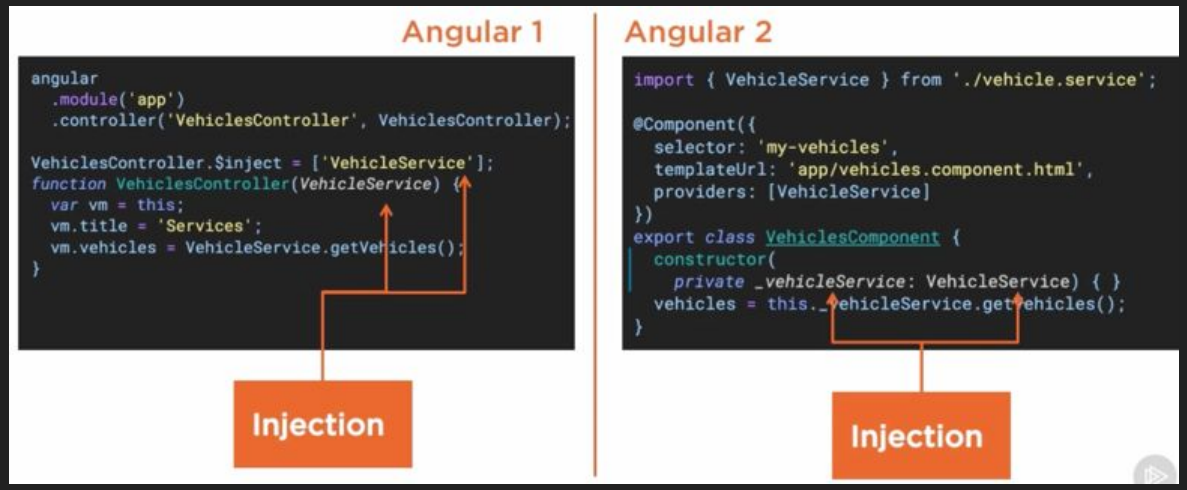
7. Injection de dépendance : Déclaration

- NG1: La déclaration se fait avec angular.service() et une chaîne de caractères qui identifie le service.
- NG2: Pas de chaîne de caractères, on déclare le service dans la propriété providers du composant qui l'utilise



7. Injection de dépendance : Injection

- NG1: Propriété .$inject qui matche les arguments passés à la fonction factory du contrôleur.
- NG2: On passe le service au constructor du composant (ou plutôt, on type un param du constructor).



Librairies de composants UI

- ng-bootstrap (https://github.com/ng-bootstrap/core)
  - Ré-écriture en Angular 2 des composants UI de Bootstrap CSS (v4).
- Angular Material (https://material.angular.io/)
  - Librairie de composants UI développés par Google spécifiquement pour Angular 2.
  - Actuellement en early alpha, mais développement assez actif.
- PrimeNG (http://www.primefaces.org/primeng/)
  - Collection de composants UI pour Angular 2 par les créateurs de PrimeFaces (une librairie populaire utilisée avec le framework JavaServer Faces).
- • Polymer (https://www.polymer-project.org/)
  - Librairie de “Web Components” extensibles par Google.
  - L’intégration avec Angular 2 est réputée simple.
- NG-Lightning (http://ng-lightning.github.io/ng-lightning/)
  - Librairie de composants et directives Angular 2 écrits directement en TypeScript sur la base du framework CSS Lightning Design System

Une question?

A vous de coder

Mais avec ngCli !

https://github.com/Giwi/angular2-beer