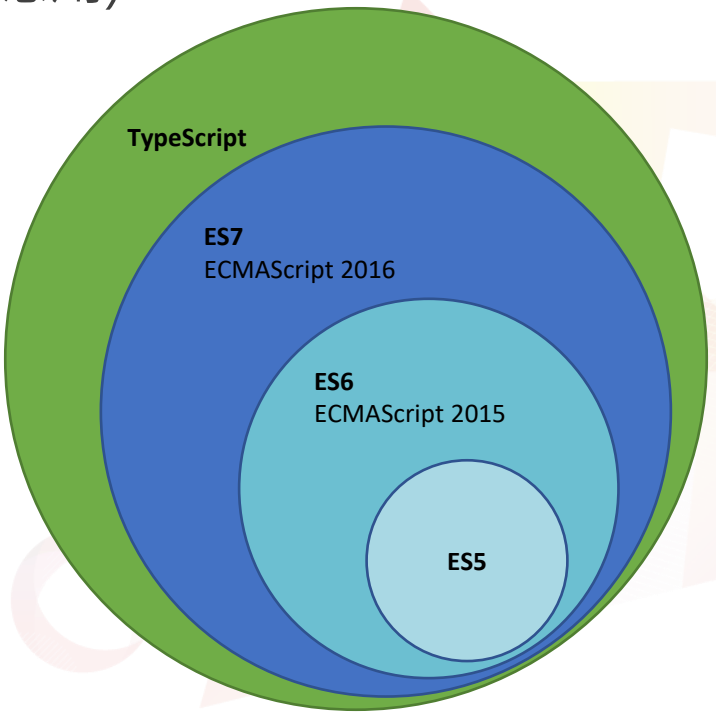
Angular基本概念

鼎新電腦
A1 商務應用雲
前端開發設計團隊

前端團隊

- SPA(Single Page Application，單頁式應用)
- 採用Typescript
- 簡潔的語法，易上手、維護



Angular 元件組成

範本 (Template)

- HTML版面配置
- 資料繫結(Binding)
- 畫面命令(Direcrtive)

類別 (Class)

- 屬性(Property)
- 建構式(Construsctor)
- 方法(Method)

中繼資料 (Metadata)

- 裝飾器(Decorator)
  - 類別
  - 屬性
  - 方法
  - 參數

Angular運作架構

模組 Module	應用程式被切分成許多「模組」
元件 Component	每個模組下有許多「元件」
樣版 Template	每個元件都可能有自己的「樣版」
中繼資料 Metadata	每個元件都可以標示「中繼資料」
資料綁定 Data Binding	樣版與元件屬性、方法可以進行綁定
宣告命令 Directive	將DOM轉換為多功能的「宣告命令」
服務 Service	由「服務」集中管理資料與運算邏輯
相依注入 DI	由「相依注入」機制管理物件生命週期

元件Component

app.component.ts

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

import

```
@Component({
selector: "app-root",
template: `
<h1>{{ title }}</h1>
`
})
```

@Component來表示為 Metadata物件
用來綁定
HTML template
CSS
Component

```
export class AppComponent {
title = " App Works! ";
}
```

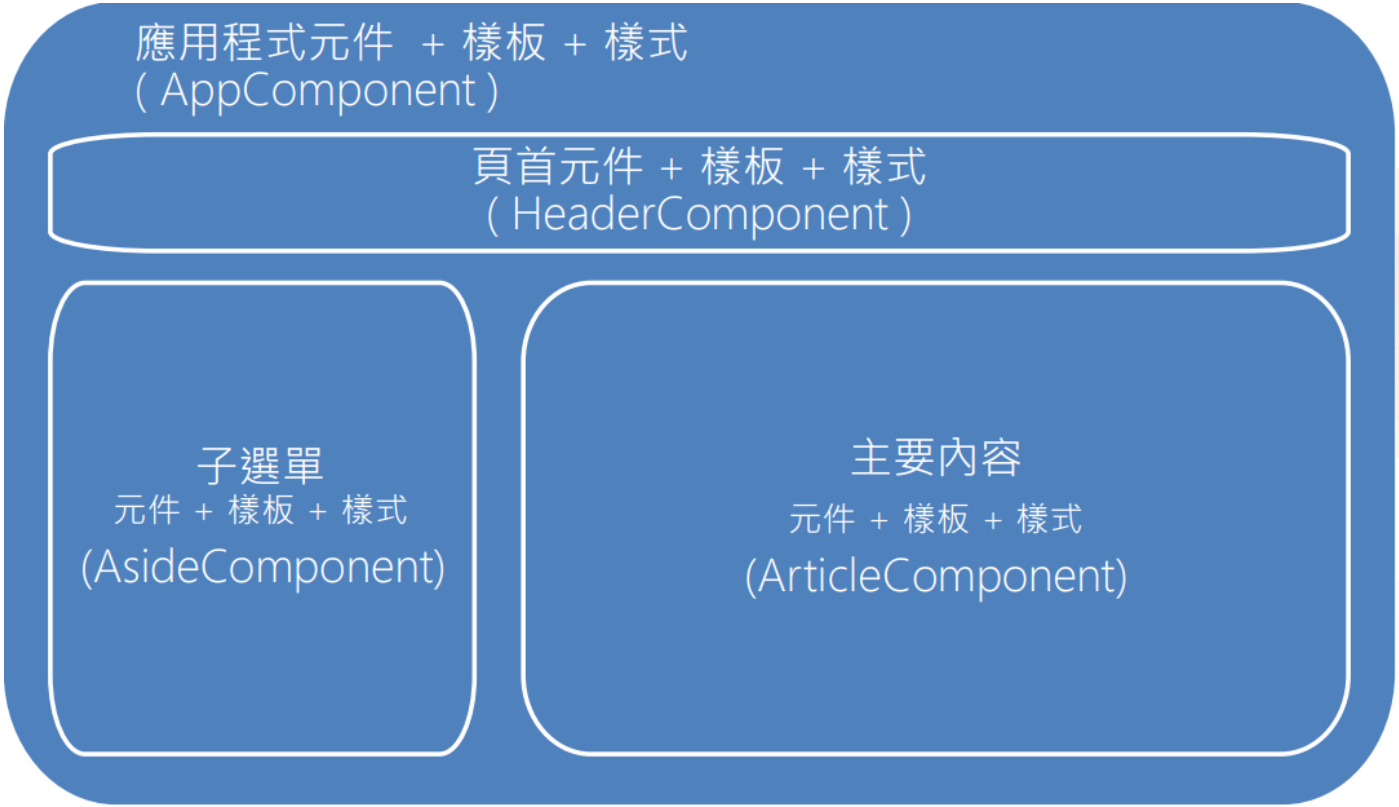
模組Module

app.module.ts

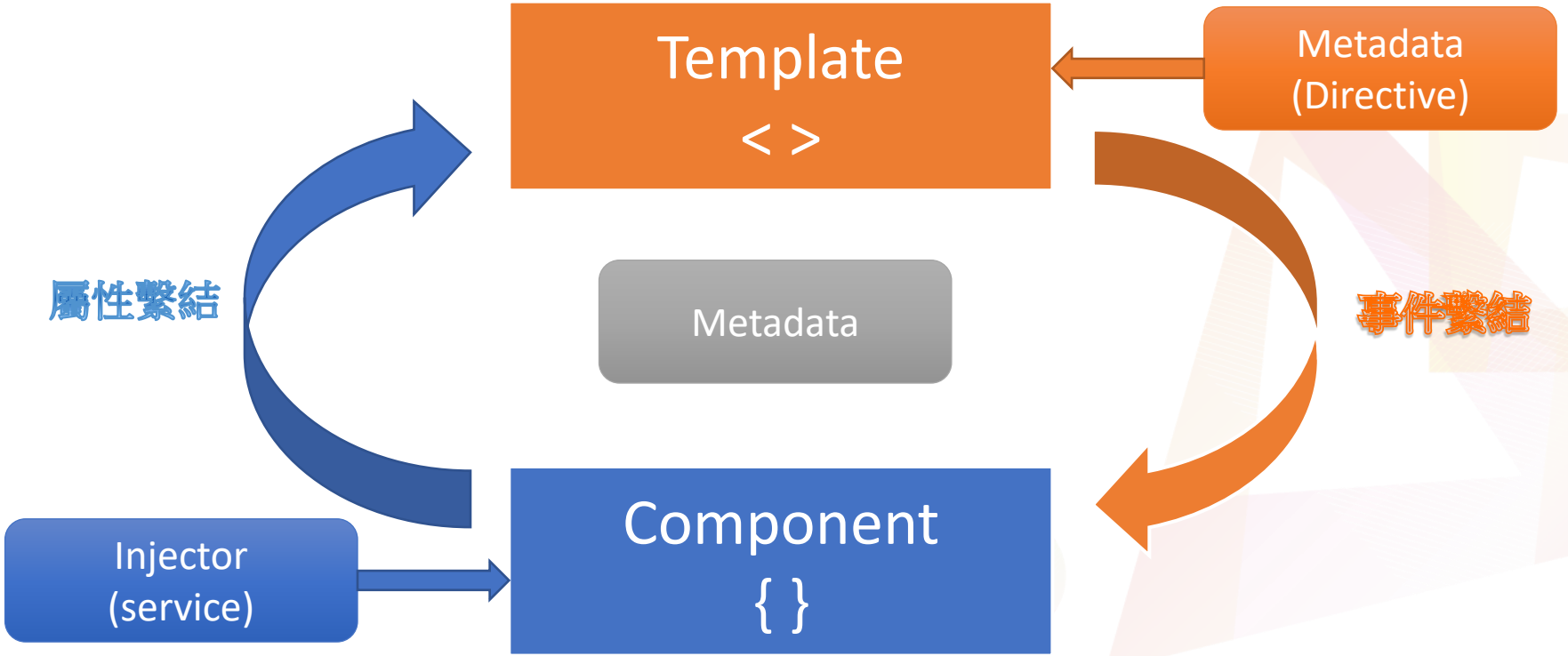
```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule({
declarations: [AppComponent],
imports: [BrowserModule],
bootstrap: [AppComponent]
})
export class AppModule { }
```

@NgModule
可包裝多個Component、Module
取起始Component

Angular 2 頁面的組成



資料繫結


Template
< >
Metadata
(Directive)
屬性繫結
Metadata
事件繫結
Injector
(service)
Component
{ }

Angular 路由機制

建立路由

- 路由設定 base href

```
<head>
 <base href="/">
 </head>
```

- 路由模組 RouterModule

```
$ ng g m [module name] --routing
```

路由策略

- 路由策略
  - PathLocationStrategy (http://localhost:4200/layout1)
  - HashLocationStrategy (http://localhost:4200/#/layout1)
- Angular路由是一個樹狀結構的元件組成
  - 路由內部還可以包含子路由
  - 每個頁面(元件)切成細個UI元件
  - 最頂端的元件為AppComponent

路由屬性

- 幾個常見的路由屬性
  - path(定義頁面名稱)
  - redirectTo (轉至另一個已定義的名稱)
  - component (對應的組件名稱)
  - canActivate (頁面驗證)

回顧重點

```
@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
  ],
  imports: [
    AppRoutingModule,
  ],
  providers: [
    { provide: LocationStrategy, useClass: HashLocationStrategy },
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

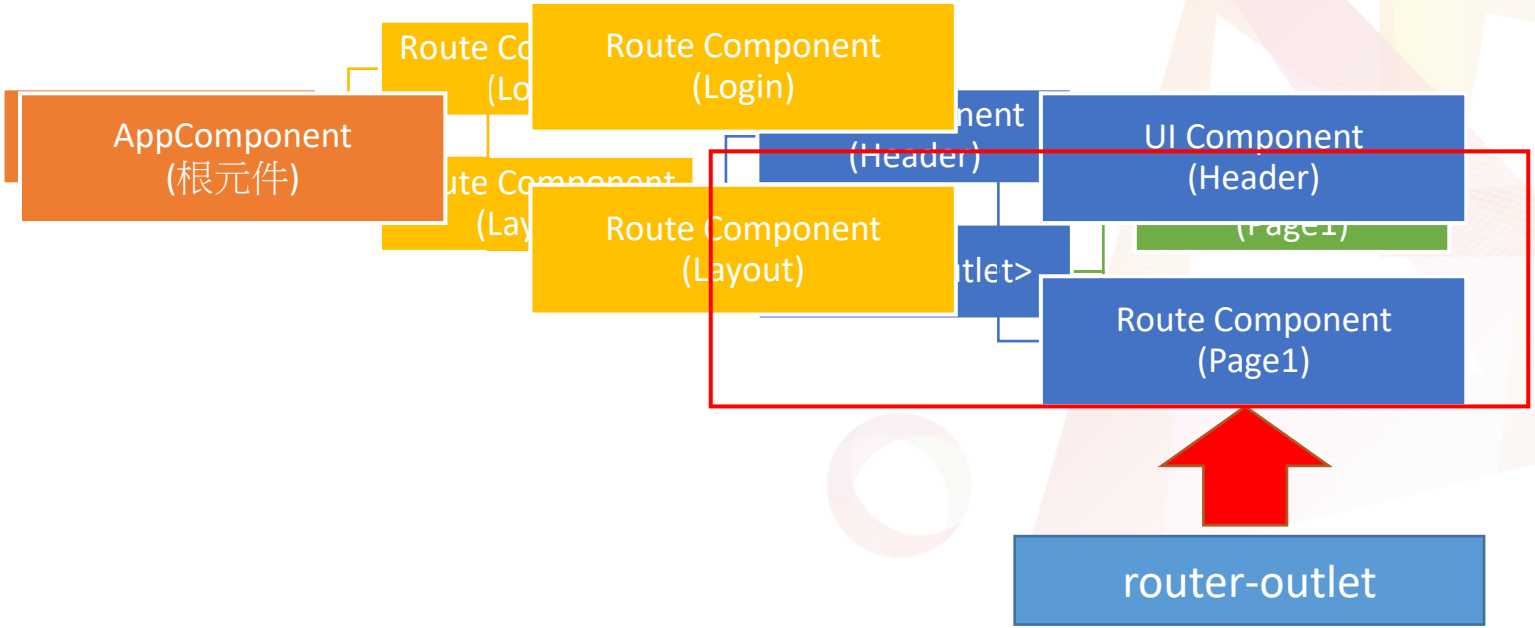
回顧重點

```
import { MobiEntryComponent } from './mobi-entry/mobi-entry.component';···

const routes: Routes = [
  { path: '', redirectTo: 'login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent, canActivate: [SignalRGuard] },
  { path: 'basic', component: BasicEntranceComponent, canActivate: [AuthGuard], data: { jobID: JOBID.GENERAL } },
  { path: '**', redirectTo: 'basic', pathMatch: 'full' }, // 萬用路由
  fallbackRoute
];

@NgModule({
  imports: [RouterModule.forRoot(routes, {
    enableTracing: false, // enableTracing:true 啟用路由追蹤
    preloadingStrategy: environment.production ? PreloadAllModules : null // PreloadAllModules預先載入
  })],
  exports: [RouterModule],
  providers: []
})
export class AppRoutingModule { }
```

RouterOutlet路由插座

路由插座

- 路由插座示意


AppComponent
(根元件)
Route Component
(Login)
UI Component
(Header)
Route Component
(Layout)
Route Component
(Page1)
router-outlet

路由插座

- 整個應用程式路由
  - 登入頁 **/login**
  - 頁面1 **/layout/page1**
  - 頁面2 **/layout/page2**

參數傳遞

參數傳遞

- 在path頁面設定參數
  - { path: 'page ', component: PageComponent}
  - **{ path: 'page/:id', component: PageDetailComponent }**

- 只要在設定在**path的路由參數**，網址路由就必須要有
- 若**沒有加上參數**，則會路由會比對成**沒有參數的page頁面**

路由連結

- routerLink
  - 在 Template 端寫死字串，適用於固定路由。

```
<a routerLink="/login">登出</a>

<a routerLink="/layout/page">作業瀏覽</a>

<a routerLink="/layout/page/3">作業3[修改] </a>
```

[路由連結]

- [routerLink]
  - 在 Template 端透過屬性的傳遞，適用於動態路由。

```
<a [routerLink]=" ['layout/page','3'] ">作業3[修改] </a>

<a [routerLink]="['layout/page',test]">作業3[修改] </a>
```

Router.navigate

- router.navigate
  - 在 Component 程式端轉換，適用於動態路由或某種事件觸發。

```
<button (click)="goWork(3)">作業3[修改] </button>
```

```
export class AppComponent {
  constructor(private router: Router) { }

  goWork(key:number) {
    this.router.navigate(['/layout/page', key]);
  }
}
```

RouterLinkActive

- Angular Router提供了routerLinkActive指令，可以用它來為選中的路由超連結添加一個CSS類別

```
<a routerLink="/layout/page" routerLinkActive="active" >
作業瀏覽</a>

<a routerLink="/layout/page/3">作業3[修改] </a>
```

Router服務元件

- 一個含有路由機制的Angular應用程式會有**Router服務元件**
  - 在任何地方都可以注入這個元件。
  - 當網址改變，Router元件或找出對應的Route物件。
- 可取路由資訊
  - config、events、navigated。
- 可透過API連結
  - navigated()、 navigatedByUrl()。

ActivatedRoute服務元件

- 表示目前正被啟用的路由物件

- 可以取得豐富路由資訊
  - Snapshot(路由快照)。
  - 透過Observabel物件來取得路由參數。

生命週期

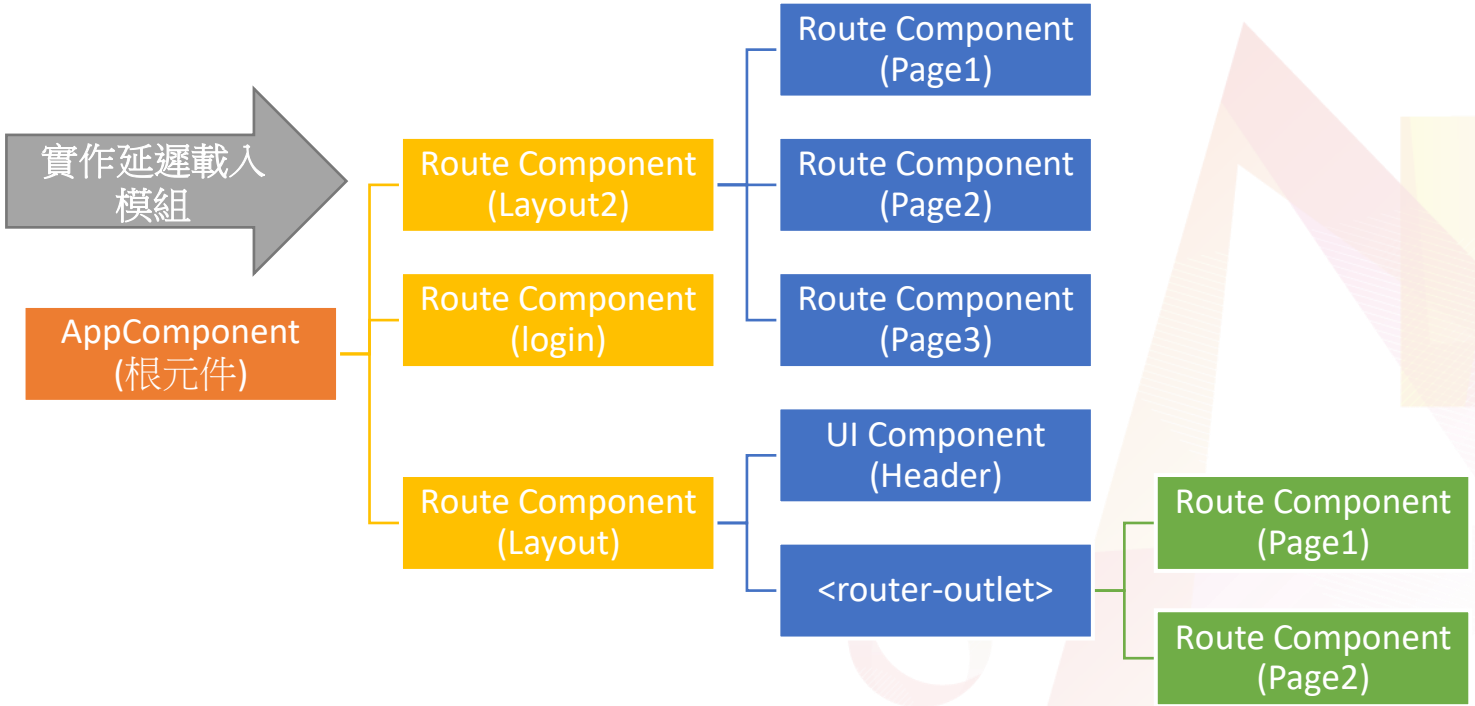
- 不同路由切換時
  - 目前的路由元件會被摧毀(Destroy)。
  - 新的路由元件會被建立(OnInit)。
- 相同的路由切換時(網址參數會被變更，但相同路由)
  - **路由沒有真的被切換，元件不會被摧毀再新增**。
  - 透過Observabel物件來取得路由參數。
  - 需訂閱服務元件ActivatedRoute來偵測變更。

延遲載入

延遲載入

- 每次只載入特定模組的程式碼
  - 可有效降低頁面下載的檔案大小。
  - 加快網頁載入速度，不會在第一次進入應用程式時就下載所有程式碼。
- 使用者進入哪個模組頁面，只下載特定功能的程式碼。
  - Angular 6 bulid時預設細分各個延遲載入模組的程式碼。
  - 大幅提升每個模組進入速度。

延遲載入


實作延遲載入模組
AppComponent (根元件)
Route Component (Layout2)
Route Component (login)
Route Component (Layout)
Route Component (Page1)
Route Component (Page2)
Route Component (Page3)
UI Component (Header)
<router-outlet>
Route Component (Page1)
Route Component (Page2)

延遲載入

- AppRoutingModule 中增加一個 Layout2的路由

- 設定 loadChildren 屬性來載入 Layout2Module

```
{ path: 'layout2', loadChildren: './layout2/layout2.module#Layout2Module'}
```

- 延遲載入的模組，不必在AppModule引入

預先載入

- 當使用者點開啟應用程式，可將其他延遲載入的程式預先下載
- 執行預先載入時，會透過非同步背景下載，不會影響使用者畫面顯示與操作
- 減少切換延遲載入模組時，需等待下載的時間

```
import { PreloadAllModules } from '@angular/router';

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

路由守門員

路由守門員

- Angular 為路由器提供了一個管理機制路由守門員(Route Guard)
  - 提供要進入、離開路由時，都可以被管控。
- Angular提供以下介面可以使用
  - **CanActivate**
  - **CanDeactivate**
  - **CanActivateChild**
  - **CanLoad**

實作 CanActivate 介面

```
import { Injectable } from '@angular/core';
import { CanActivate, Router, RouterStateSnapshot, ActivatedRouteSnapshot } from
'@angular/router';
import { Identity } from './identity';
@Injectable()
export class AuthGuard implements CanActivate {
  constructor(private router: Router) { }
  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    if (!Identity.IsLogin) {
      this.router.navigate(['/login']);
      return false;
    }
    return true;
  }
}
```

```
{ path: 'page', component: PageComponent, canActivate: [AuthGuard] }
```

感謝