



«AngularJS 開發實戰»

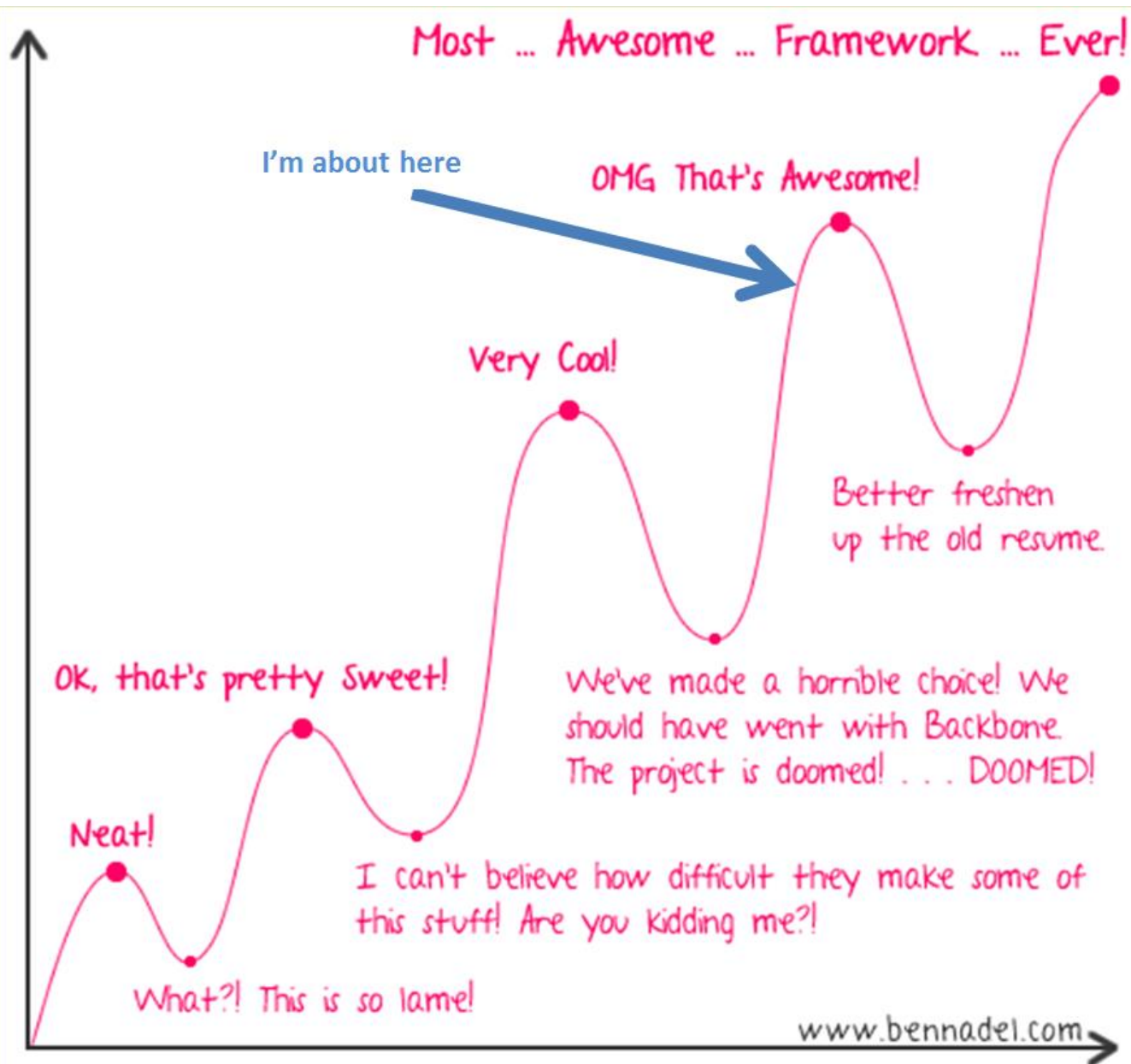
模組開發篇



多奇數位創意有限公司

技術總監 黃保翕 (Will 保哥)

部落格：<http://blog.miniasp.com/>



My Feelings About AngularJS Over Time

Core Concepts

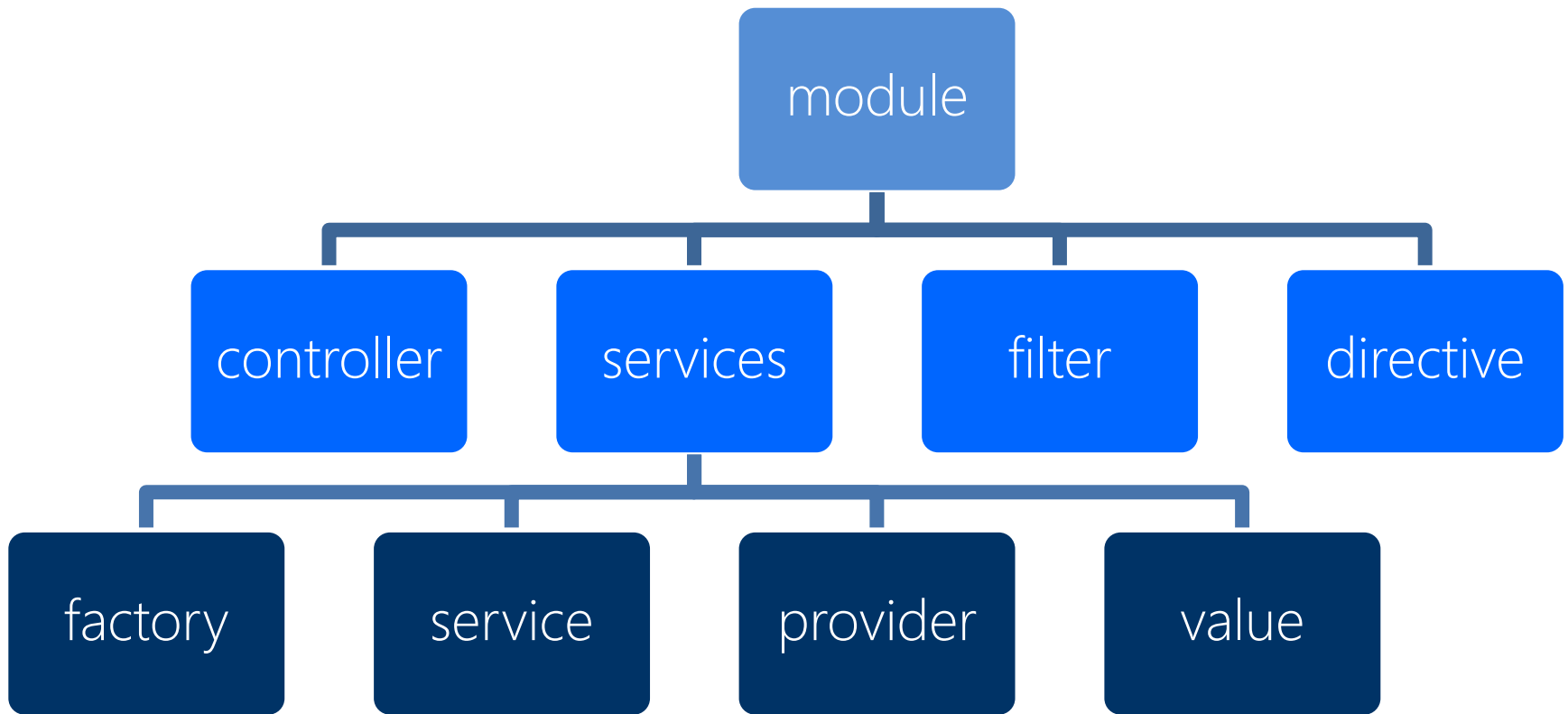
ANGULARJS 核心觀念



AngularJS 主要元件組成

- 控制器 (Controllers)
 - 控制器是 AngularJS 裡面最重要的元件
 - 大多 AngularJS 程式碼的執行起點、包含 UI 邏輯與狀態處理
- 服務 (Services)
 - 複雜的商業邏輯、應用程式狀態保存
 - 如果要跟後端程式溝通，大多也應寫在服務裡
- 過濾器 (Filters)
 - 服務的一種，用來過濾、篩選、轉換資料之用
- 畫面與指令 (Views / Directives)
 - 所有的 Directives 都屬於這一部份
 - 顯示資料與使用者互動、收集網頁事件、雙向資料繫結

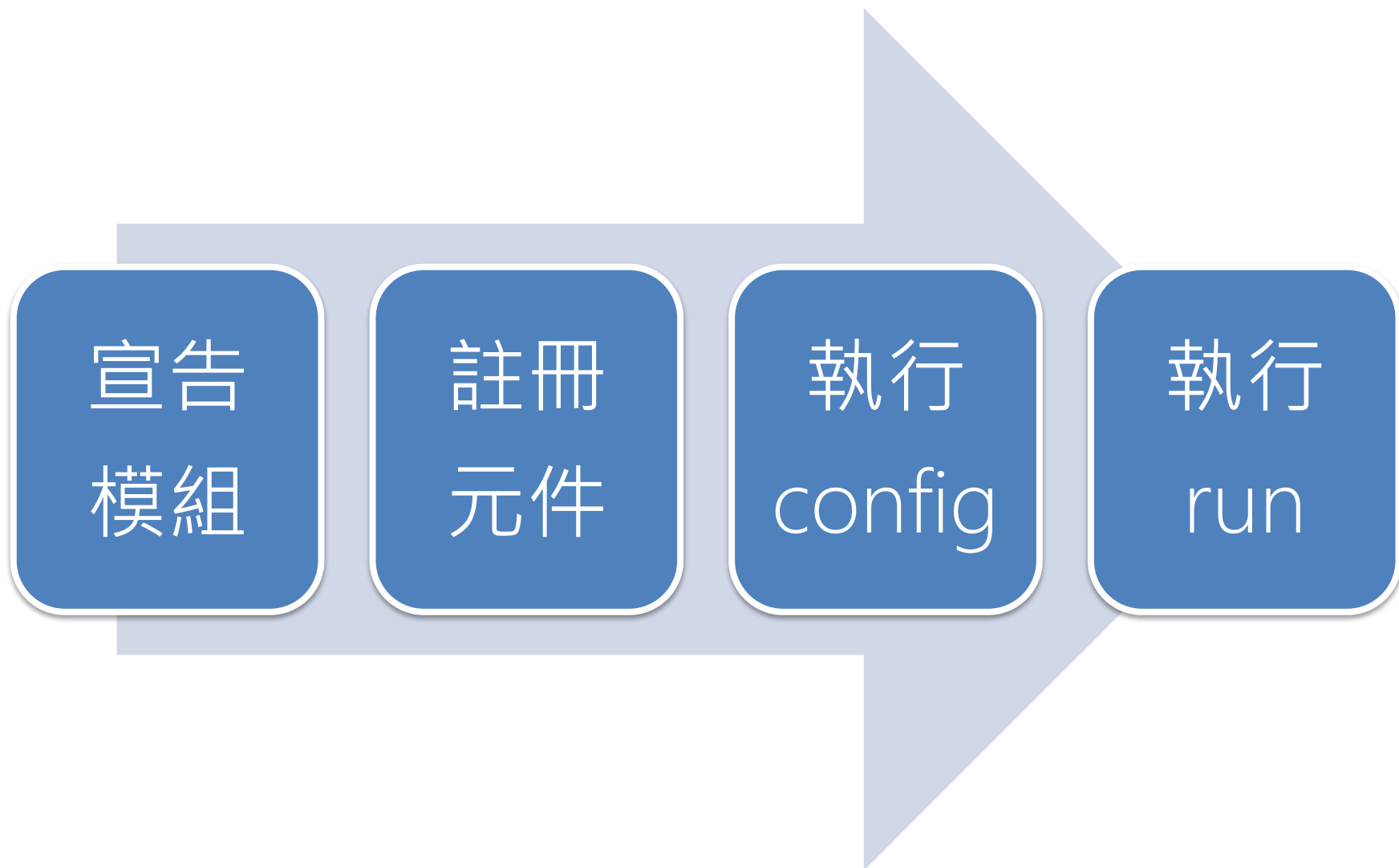
認識 Angular 元件模組化架構



使用 Angular 模組

- 建立模組、載入相依模組、取得模組 ([angular.module](#))
 - `var myModule = angular.module('myModule', []);`
 - `var myModule = angular.module('myModule', ['ngTouch']);`
 - `var myModule = angular.module('myModule');`
- 註冊服務 ([angular.Module](#))
 - `myModule.controller('name', [ControllerFn]);`
 - `myModule.value('name', 'any value');`
- 模組初始化 (設定提供者參數) ([Providers](#))
 - `myModule.config(['$providerName', ConfigFn]);`
- 模組初始化 (設定服務實體) ([Modules](#))
 - `myModule.run(['$instanceName', RunFn]);`

Angular 模組的載入與啟動順序



```
angular.module('myApp', ['Sub1', 'Sub2'])
```

宣告與註冊的順序不重要

宣告模組與
註冊元件

Sub1

Sub2

myApp

執行
config

Sub1

Sub2

myApp

執行 run

Sub1

Sub2

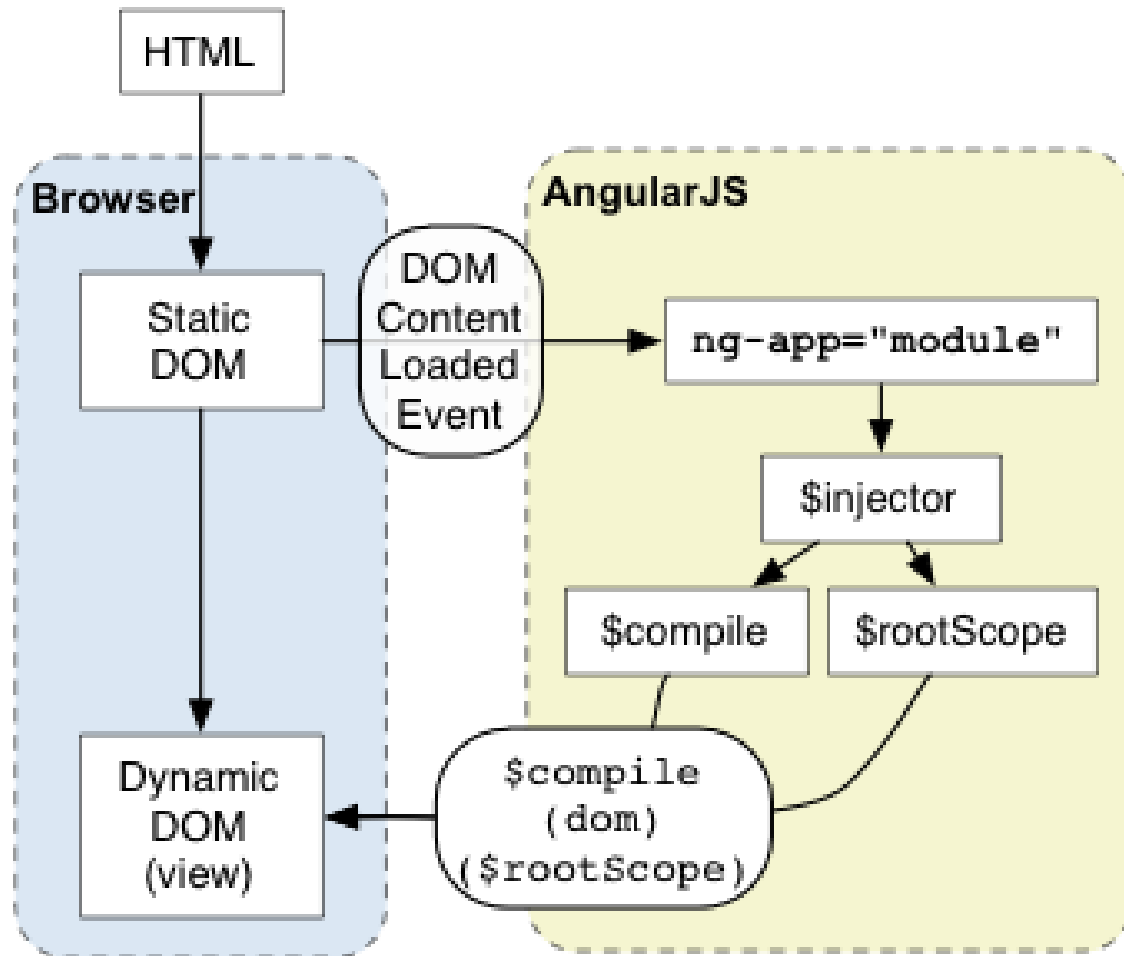
myApp

DEMO: <http://jsbin.com/nujufa/edit?html,console,output>

練習：使用 AngularJS 模組

- 將[此專案](#)改用 angular.module 建立自訂模組
 - 設定 [\\$controllerProvider](#) 讓 NG 允許**全域控制器**宣告
 - `$controllerProvider.allowGlobals();`
 - 改用 `myModule.controller()` 明確宣告 Angular 控制器
- 在自訂模組中載入 [ngCookies](#) 模組
 - 記得要載入必要的 **angular-cookies.js**
 - 在 `MainCtrl` 控制器注入 `$cookies` 服務
 - 設定 `cookiesProvider` 服務提供者
 - `$cookiesProvider.defaults.path = '/';`
- 建立共用變數、注入控制器、顯示於頁面
 - `myModule.value('version', '1.0.0');`

AngularJS 的啟動過程 ([Bootstrap](#))



AngularJS 的啟動過程 (Bootstrap)

- 自動啟動

- 偵測 DOMContentLoaded 事件
- 尋找 ng-app 指令 (directive) 並取得啟動模組名稱
- 建立 **\$injector** 注入器物件 (每個應用程式只會有一個)
- 編譯 DOM 物件 (視 ng-app 所在元素為根元素)

```
<!doctype html>
<html ng-app="optionalModuleName" ng-strict-di>
  <body>
    I can add: {{ 1+2 }}.
    <script src="angular.js"></script>
  </body>
</html>
```

AngularJS 的啟動過程 (Bootstrap)

- **手動啟動**

- 必須等待所有模組載入網頁後才能啟動
- 執行 `angular.bootstrap()` 即可啟動 Angular 應用程式
 - 先找到 DOM 物件，並傳入為第一個參數
 - 決定要啟動的主要模組名稱
 - 編譯 DOM 物件 (視 `ng-app` 所在元素為根元素)
- 注意事項
 - 啟動 Angular 應用程式後就不能再加入 `controllers`, `services`, `directives`, ... 等元件/服務
 - 採用手動初始化時，不能同時使用 `ng-app` 指令
- 使用時機
 - 非同步載入大量 JavaScript 時可採用手動初始化

AngularJS 的啟動過程 (Bootstrap)

```
<!doctype html>
<html>
<body>
  <div ng-controller="MyController">
    Hello {{greetMe}}!
  </div>
  <script src="http://code.angularjs.org/snapshot/angular.js"></script>

  <script>
    angular.module('myApp', [])
      .controller('MyController', ['$scope', function ($scope) {
        $scope.greetMe = 'World';
      }]);

    angular.element(document).ready(function() {
      angular.bootstrap(document, ['myApp']);
    });
  </script>
</body>
</html>
```

練習：改用手動啟動 AngularJS 模組

- 將所有 AngularJS 程式碼都移至 app.js
- 移除 ng-app="myApp" 屬性
- 改用**手動**方式啟動 AngularJS 模組

AngularJS 的啟動過程 (Bootstrap)

- 延遲啟動

- 如果 window.name 的值為 "NG_DEFER_BOOTSTRAP!" 開頭，當 angular.bootstrap() 執行時會暫停執行，並等到 angular.resumeBootstrap() 被執行時才會繼續。

- 使用時機

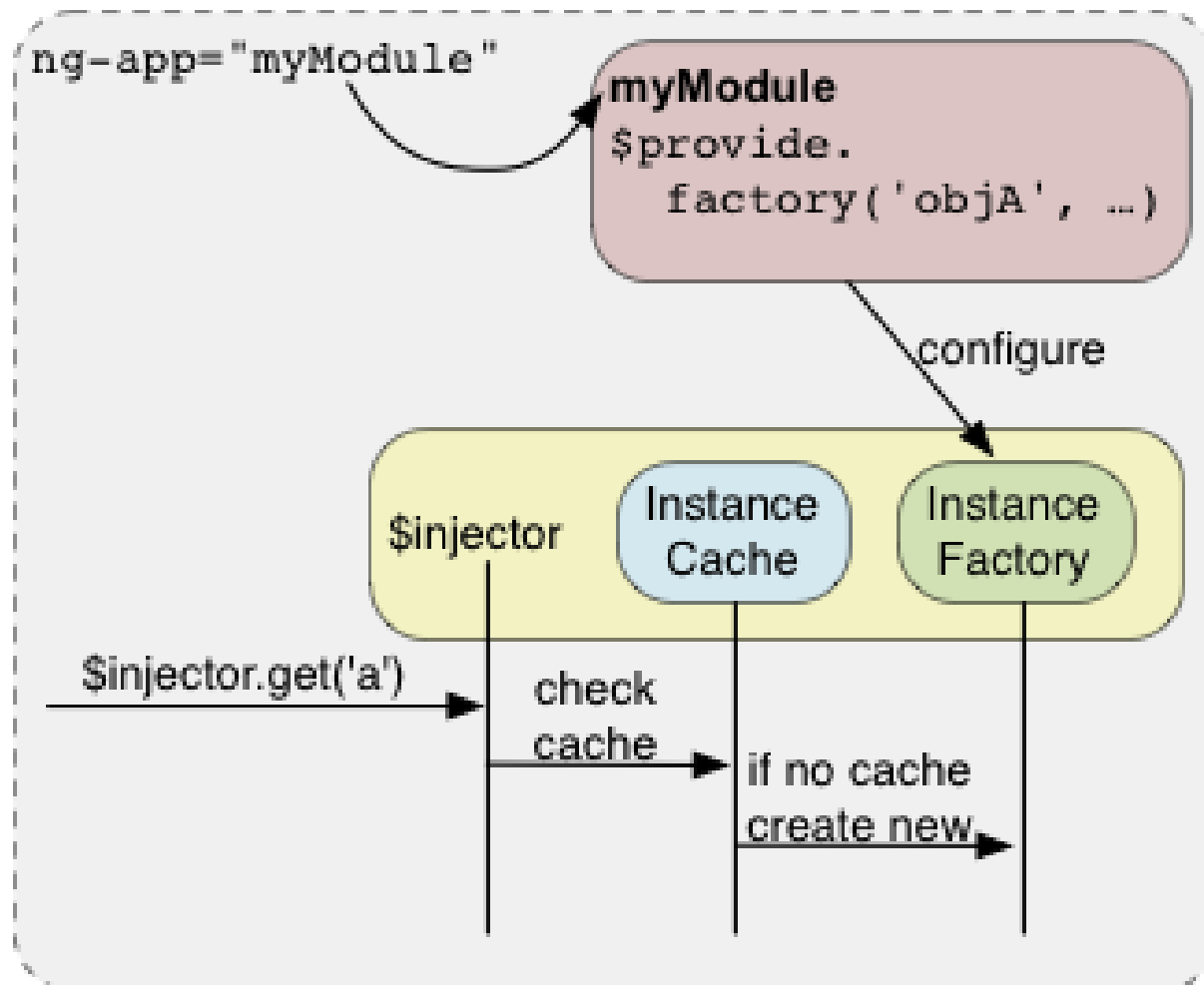
- 撰寫 Chrome 擴充套件時，最適合用這個功能
- 擴充套件範例：
 - [AngularJS Batarang](#) v0.8.5
AngularJS WebInspector Extension for Chrome
<https://github.com/angular/angularjs-batarang>
 - [AngularJS Batarang \(Stable\)](#) v0.4.3 [如果新版有 Bug 才安裝]

介紹 \$injector 注入器物件

- 特性
 - Angular 應用程式，只會有一個 \$injector 注入器物件
- 用途
 - \$injector 注入器物件主要用來取得由 Provider 建立的服務、負責**實體化**型別、執行特定方法或載入其他模組
- 建立獨立的 \$injector 物件
 - `var $injector = angular.injector();`
 - `var $injector2 = $injector.get('$injector');`
 - `var $injector3 = $injector.invoke(function($injector) {
 return $injector;
});`

<http://jsbin.com/bamayo/edit?html,console,output>

模組 (Modules) 與 注入器 (Injector)



模組 (Modules) 與 注入器 (Injector)

- 注入器 (Injector)
 - 是一個 *service* locator
 - 一個 ng-app 只會有一個 injector
 - injector 負責維護一群**內部物件**快取
 - 每一個註冊在 injector 的物件都會有個**名稱**
 - 如果透過**名稱**找不到物件，會透過 instance factory 自動建立
- 模組 (Modules)
 - 一個設定**注入器** instance factory 的管道
 - [http://docs.angularjs.org/api/AUTO.\\$provide](http://docs.angularjs.org/api/AUTO.$provide)

使用 \$injector 注入器物件

- 建立 ng 模組的 \$injector 注入器物件
 - `var $injector = angular.injector(['ng']);`
 - `var $http = $injector.get('$http');`
- 建立自訂模組下的 \$injector 注入器物件
 - `var $injector = angular.injector(['ng', 'myApp']);`
 - `var version = $injector.get('version');`
- 透過 DOM 直接取得該範圍內的 \$injector 注入器物件
 - `var $injector = angular.element('#main').injector();`
 - `var $injector = angular.element(elm).injector();`
 - `var version = $injector.get('version');`

練習：從 jQuery 取得 \$injector 物件

- 撰寫一個 jQuery 事件

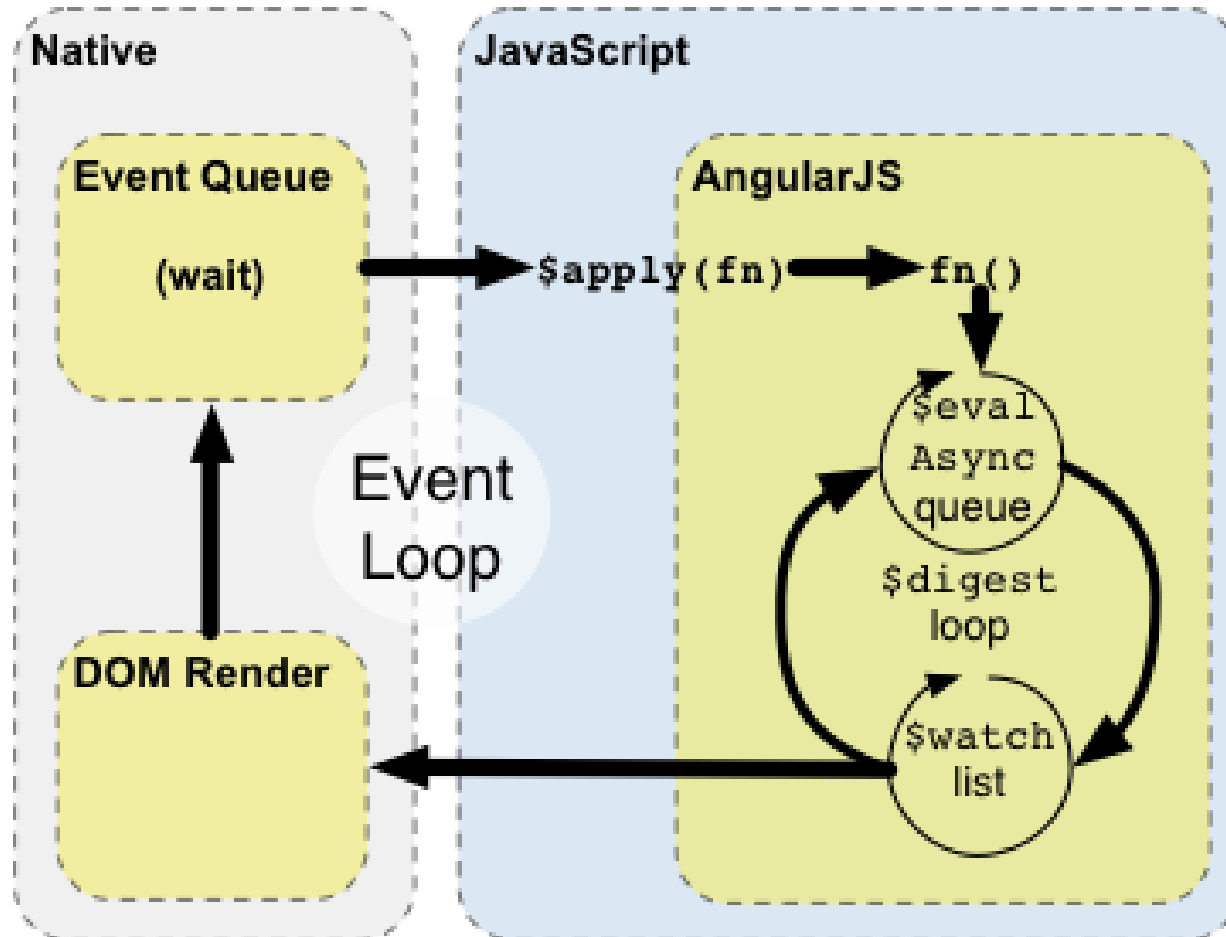
```
<button onclick="run()">
```

取得 version 值

```
</button>
```

- 從事件中取得 AngularJS 模組中的 \$injector 物件
- 透過 \$injector 取得 AngularJS 運行中的資料

Angular 執行生命週期 (browser event loop)



Angular 執行生命週期 (browser event loop)

- 事件觸發執行 scope.\$apply(stimulusFn)
- stimulusFn 被執行，用以修改應用程式狀態
- Angular 進入 \$digest 迴圈，該迴圈中還有兩個小迴圈
 - \$evalAsync queue 用於非同步執行部分工作
 - \$watch list 包含一組 expressions 執行時會偵測模型變動
- \$digest 迴圈會**不斷重複執行，直到 model 狀態穩定**
 - 直到 \$evalAsync queue 變成空的
 - 直到 \$watch list 不再偵測到任何變化
- 當 **\$digest** 迴圈結束後
 - 依據偵測到的變更交由瀏覽器更新 DOM 物件

Angular 執行生命週期 (browser event loop)

- 實際範例：<http://jsbin.com/ragifa/edit?html,output>
 - 編譯時期
 - `<input type="text" ng-model="main.Name">`
[ng-model](#) 在 [input directive](#) 設立一個 `keydown` 事件
 - `{{main.Name}}`
經由 [\\$interpolation](#) 自動設立一個 [\\$watch](#) 監視 `main.Name` 的變更 (意即加入到 `watch list` 裡面)
 - 執行時期
 - 在 `input` 欄位上輸入任意一個字元 `x` 並觸發瀏覽器的 `keydown` 事件
 - [input directive](#) 呼叫 `$apply("name = 'X';")` 並更新模型
 - Angular 套用 `name = 'X';`
 - [\\$digest](#) 迴圈開始執行
 - `$watch list` 偵測到 `name` 發生變化並通知 `$interpolation` 進行DOM更新
 - Angular 離開執行環境與瀏覽器的 `keydown` 事件，並重繪網頁

計算 Watch 清單數量與 Digest 迴圈消化時間

- ng-stats
 - Utility to show stats about angular digest/watches.
 - <http://angular-js.in/ng-stats/>
- 安裝
 - bower install ng-stats --save
 - gulp bower
- 範例 (放進本機執行)
 - <https://gist.github.com/doggy8088/4244a6942f31ba79bb7c>
 - <http://jsbin.com/xiqoni/edit>
- 任務
 - 檢查 watches 的數量與 digest 執行速度

關於 Scope 範圍物件

- 用來偵測 Model 物件的變更
 - ng 靠 Scope 連結 View 與 Controller 之間
 - View: 表達式 (expression)
 - Controller: `$scope`
- Scope 擁有物件繼承特性
 - 類似 DOM 的樹狀結構
 - 有些 directives 會建立新的 Scope
 - `ng-controller`, `ng-repeat`, `ng-switch`
 - `ng-view`, `ng-include`
 - 其他自訂的 directives

了解 Scope 範圍物件

- 什麼是 Scope 呢？
 - 應用程式的 model 物件
 - Angular Expression 的執行環境 (execution context)
 - 具有巢狀結構 (多階層 Scope 物件)
 - 可以監視 Expression 變更與傳遞事件
- Scopes 的特性
 - Scopes 提供 [\\$watch](#) API 可觀察 model 變動
 - Scopes 提供 [\\$apply](#) API 可傳遞 model 變動給 Views
 - Scopes 是擁有巢狀結構，但有兩種類型：
 - child scopes 會自動從**上層範圍**繼承資料下來
 - [isolate scopes](#) 獨自存在的 Scope 不會從**上層範圍**繼承物件
 - Scopes 將與 Views 裡面的 Expression 緊緊相依
 - Scopes 是將 controller 與 views 連接在一起的工具！

顯示 Scope 範圍物件的區域

```
<style type="text/css" media="screen">
    .ng-scope {
        border: 1px solid red;
    }
</style>
```

```
<!DOCTYPE html>
<html ng-app class="ng-scope">
  <head>...</head>
  <body>
    <div>
      <div ng-controller="GreetCtrl" class="ng-scope ng-binding">
        Hello World!
      </div>
      <div ng-controller="ListCtrl" class="ng-scope">
        <ol>
          <!-- ngRepeat: name in names -->
          <li ng-repeat="name in names" class="ng-scope ng-binding">
            Igor
          </li>
          <li ng-repeat="name in names" class="ng-scope ng-binding">
            Misko
          </li>
          <li ng-repeat="name in names" class="ng-scope ng-binding">
            Vojta
          </li>
        </ol>
      </div>
    </div>
  </body>
</html>
```

Scope 物件的繼承特性

- \$rootScope
 - 最高層級的 Scope 物件
 - 可透過 DI 注入到 controllers 裡面
- \$scope
 - 隸屬於「特定」層級的 Scope 物件
- 安裝工具
 - [ng-inspector for AngularJS](http://ng-inspector.org/) (<http://ng-inspector.org/>)
 - [Angular Scope Inspector](#)
 - [AngularJS Batarang](#) v0.8.5
 - [AngularJS Batarang \(Stable\)](#) v0.4.3
- 練習查看 Scope 物件：<http://output.jsbin.com/yaqozu>

練習：AngularJS 根範圍 (\$rootScope)

- JavaScript

```
angular.module('app', [])  
  .run(function($rootScope){  
    $rootScope.names = [];  
  });
```

- 範例程式

- 在所有預設作用域下的變數都在根物件下
- <http://jsbin.com/iyiviv/1/edit>

練習：AngularJS 子範圍 (\$scope)

- JavaScript

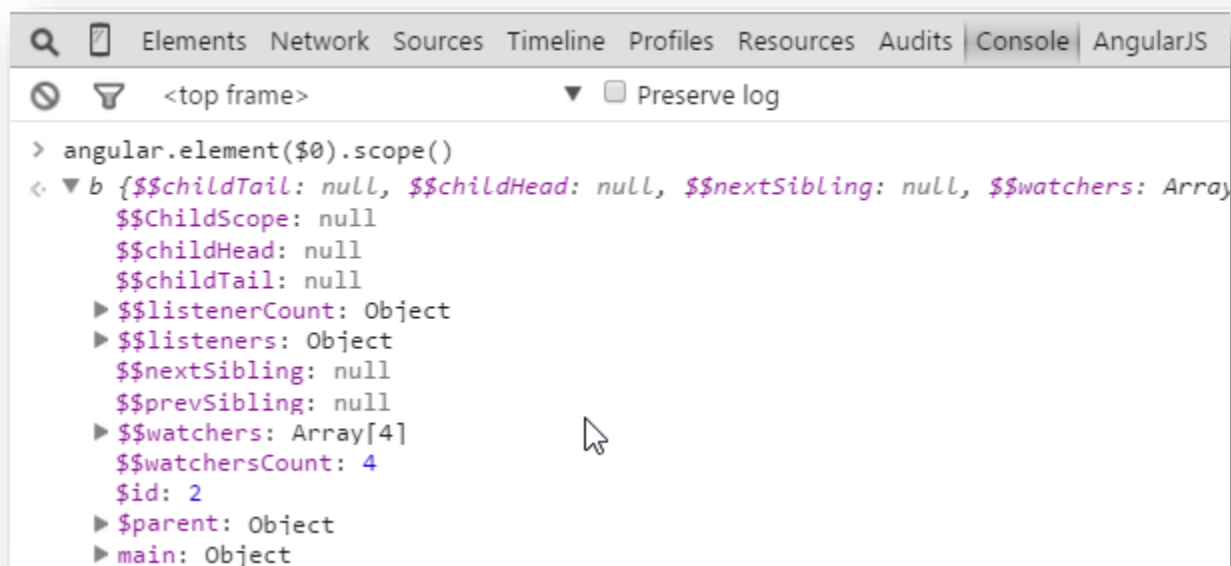
```
angular.module('app', []);  
.run(function($rootScope){  
    $rootScope.name = "James";  
});  
.controller("MyController", function ($scope) {  
    $scope.name = "Will";  
}  
);
```

- 範例程式

- 子作用域下的屬性都會有「原型繼承」特性
- <http://jsbin.com/quwari/1/edit>

如何從 DOM 取得 Scopes 物件

- 範例程式
 - `var currentScope = angular.element('.id').scope();`
- 在 Chrome 開發者工具中偵錯
 - 透過 `$0` 取得目前選到的 DOM 物件
 - `var currentScope = angular.element($0).scope();`



練習：透過 jQuery 取得 AngularJS 範圍

- 撰寫一個 jQuery 事件

```
<button onclick="run()">
```

取得 `$scope` 模型

```
</button>
```

- 從事件中取得 AngularJS 模組中特定 ng-controller 內的 `$scope` 物件
- 目的
 - 在 Angular 生命週期外取得 Angular 物件

Scope 事件傳遞

- 註冊 Scope 事件 ([\\$rootScope.Scope](#))
 - 透過 DI 注入 `$scope` 服務
 - 使用 `$scope.$on('eventName', eventFn)`; 註冊事件
 - 使用 `$scope.$emit(name, args)`; 向**上層**觸發事件
 - 使用 `$scope.$broadcast(name, args)`; 向**下層**觸發事件
- 範例程式
 - <http://jsbin.com/luvizi/edit?html,js,output>
- 注意事項
 - 一定要注入 `$scope` 服務，不能用 `controllerAs` 語法

Scope 生命週期

1. 建立 Scope 物件 (Creation)

- Angular 應用程式啟動時，預設會透過 `$injector` 建立 `$rootScope` 實體，其他 directives 也可能會建立各自的 `$scope`

2. 監視器註冊 (Watcher registration)

- 在 directives 連結到 templates 的時候，會在相對應的 Scope 上建立監視器，用以連結 View 與 Models

3. 模型變更 (Model mutation)

- 所有的模型變更動作，都應該在 `scope.$apply()` 底下完成
- Angular API 預設會自動完成這件事，所以在 controller 裡不需要再執行額外的 `scope.$apply()` 動作

4. 監視變更 (Mutation observation)

- `$apply` 執行完畢後，會先引發 `$rootScope.$digest` 迴圈，並且所有 child scopes 的 `$scope` 也進行 `$digest` 迴圈，所有註冊過的監視器(watcher)都會再次檢查是否引發模型變更，如果是，就會在所有監視器執行完後，再次執行一次 `$digest` 迴圈。

5. 消滅 Scope 物件 (Scope destruction)

- 當 child scopes 已經不再使用時，會透過 [`scope.\$destroy\(\)`](#) 摧毀所有 `$scope` 物件實體 (會釋放記憶體與啟動垃圾回收機制)，未來的 `$digest` 迴圈也將不再於這個 `$scope` 執行。


Scopes 與 Directives 的關係

- 在 Angular 執行生命週期的「編譯時期」，在編譯 DOM 範本時 directives 通常會區分兩類：
 - 觀察 directives
 - 透過 [\\$watch\(\)](#) 註冊監視器，監視所有 Angular Expression 並在模型發生變更時通知更新 view
 - 監聽 directives
 - 例如 ng-click 事件
 - 在 DOM 註冊相對應的事件監聽器(listener)，當 DOM 的事件被觸發，則透過 [\\$apply\(\)](#) 觸發 digest 迴圈並更新 views
- 注意事項
 - 如果是在 Angular 以外的事件被觸發，一定要手動執行 [\\$apply\(\)](#) 才會執行 digest 迴圈 (例如 jQuery UI)

練習：混合 AngularJS 與 jQuery 使用

- 混合 AngularJS 與 jQuery 使用
 - <http://jsbin.com/angular-jquery-mixed/1/edit>
- 結合 AngularJS 與 jQuery UI
 - <http://jsbin.com/angular-jquery-mixed/2/edit>
- 結合 AngularJS 與 jQuery UI (修正繫結問題)
 - <http://jsbin.com/angular-jquery-mixed/3/edit>

Directives 不一定會建立 Scopes



The screenshot shows the AngularJS documentation for the `ngController` directive. On the left is a sidebar with a list of directives, where `ngController` is highlighted in red. The main content area has the title `ngController` with the subtitle `- directive in module ng`. The text explains that the `ngController` directive attaches a controller to a scope and supports MVC principles. It lists the MVC components: Model, View, and Controller. A note mentions that controllers can also be attached to the DOM using `ng-controller` in the template. At the bottom, the 'Directive Info' section states that this directive creates a new scope and executes at priority level 500.

ngBindHtml
ngBindTemplate
ngBlur
ngChange
ngChecked
ngClass
ngClassEven
ngClassOdd
ngClick
ngCloak
ngController
ngCopy
ngCsp
ngCut
ngDbclick
ngDisabled
ngFocus
ngForm
ngHide
ngHref
ngIf
ngInclude
ngInit
ngJq
ngKeydown

ngController

- directive in module ng

The `ngController` directive attaches a controller to a scope and supports the principles behind the Model-View-Controller (MVC) pattern.

MVC components in angular:

- Model — Models are the properties of a scope; scopes are created by the `ngScope` directive.
- View — The template (HTML with data bindings) that is rendered by the `ngView` directive.
- Controller — The `ngController` directive specifies a Controller to decorate the scope with functions and values.

Note that you can also attach controllers to the DOM by declaring a controller on the DOM element using `ng-controller` in the template.

Directive Info

This directive creates new scope.
This directive executes at priority level 500.

Scopes 與 Controllers 的關係

- 密切的合作關係
 - controller 透過 **\$scope** 傳遞 methods 到 views 裡
 - controller 定義的 methods 可以修改 **\$scope** 的屬性內容
 - controller 可以定義額外的 [watches](#) 監視特定模型，這些監視器 (watches) 會在 methods 執行完後立刻執行

```
$scope.$watch('name', function(newValue, oldValue) {  
    $scope.counter = $scope.counter + 1;  
});
```

\$scope 與 controllerAs 語法

- Angular 1.1 以前

```
app.controller('MainCtrl', function ($scope) {  
    $scope.title = 'Some title';  
});  
<div ng-controller="MainCtrl">  
    {{ title }}  
</div>
```

- Angular 1.2 以後

```
app.controller('MainCtrl', function () {  
    var vm = this;  
    vm.title = 'Some title';  
});  
<div ng-controller="MainCtrl as main">  
    {{ main.title }}  
</div>
```

\$scope 與 controllerAs 語法

<http://toddmotto.com/digging-into-angulars-controller-as-syntax/>

```
<div ng-controller="MainCtrl">
  {{ title }}
  <div ng-controller="AnotherCtrl">
    Scope title: {{ title }}
    Parent title: {{ $parent.title }}
    <div ng-controller="YetAnotherCtrl">
      {{ title }}
      Parent title: {{ $parent.title }}
      Parent parent title: {{ $parent.$parent.title }}
    </div>
  </div>
</div>
```

```
<div ng-controller="MainCtrl as main">
  {{ main.title }}
  <div ng-controller="AnotherCtrl as another">
    Scope title: {{ another.title }}
    Parent title: {{ main.title }}
    <div ng-controller="YetAnotherCtrl as yet">
      Scope title: {{ yet.title }}
      Parent title: {{ another.title }}
      Parent parent title: {{ main.title }}
    </div>
  </div>
</div>
```


使用 \$watch 的注意事項 (效能考量)

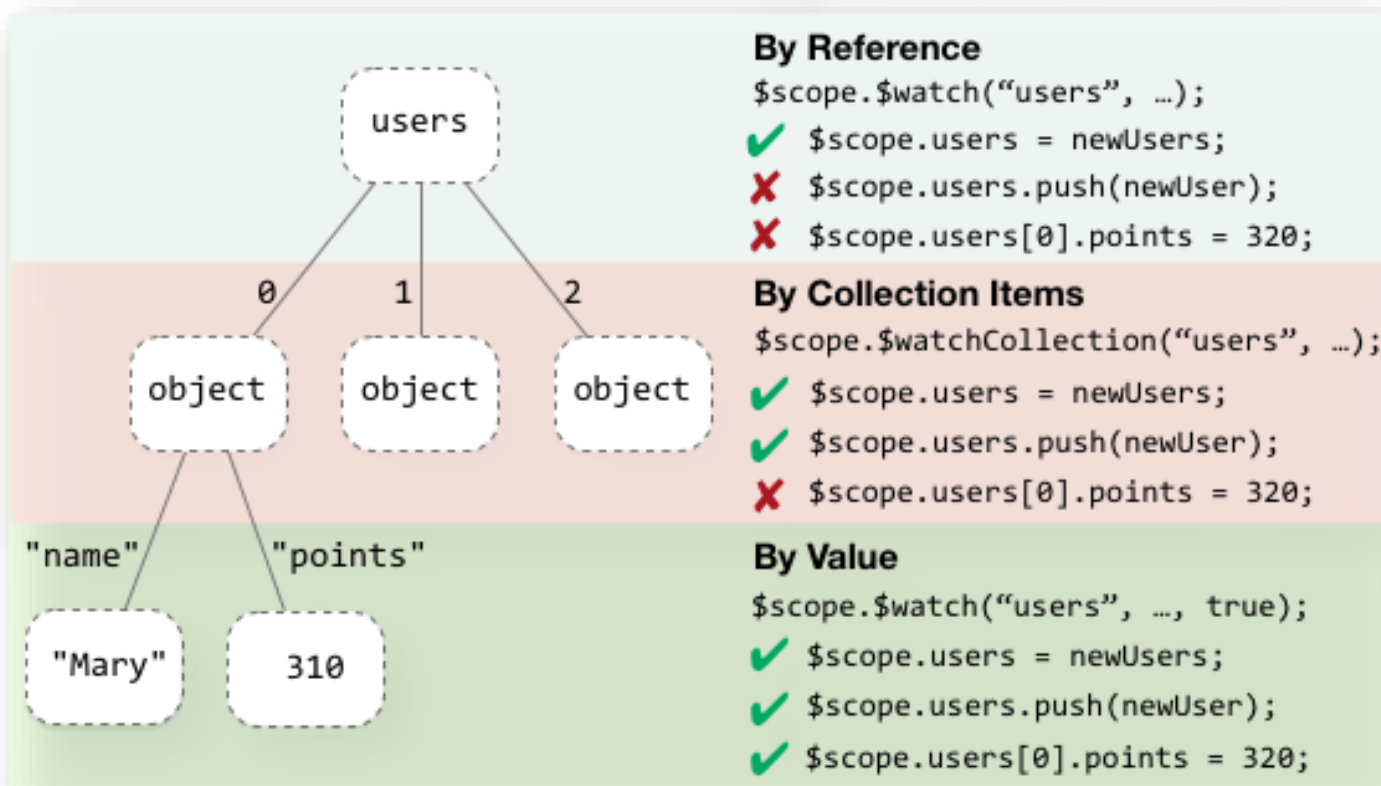
- 頻繁檢查 (Dirty Check)
 - 參考 [Angular 執行生命週期 \(browser event loop\)](#)
 - 由於一個頁面中的 watches 可能很多，而 Angular 的頻繁檢查特性會導致透過 [\\$watch](#) 註冊的 listener 會執行很多次，因此切記不要在該 listener 中操作 DOM 物件。

使用 \$watch 的注意事項 (監視深度)

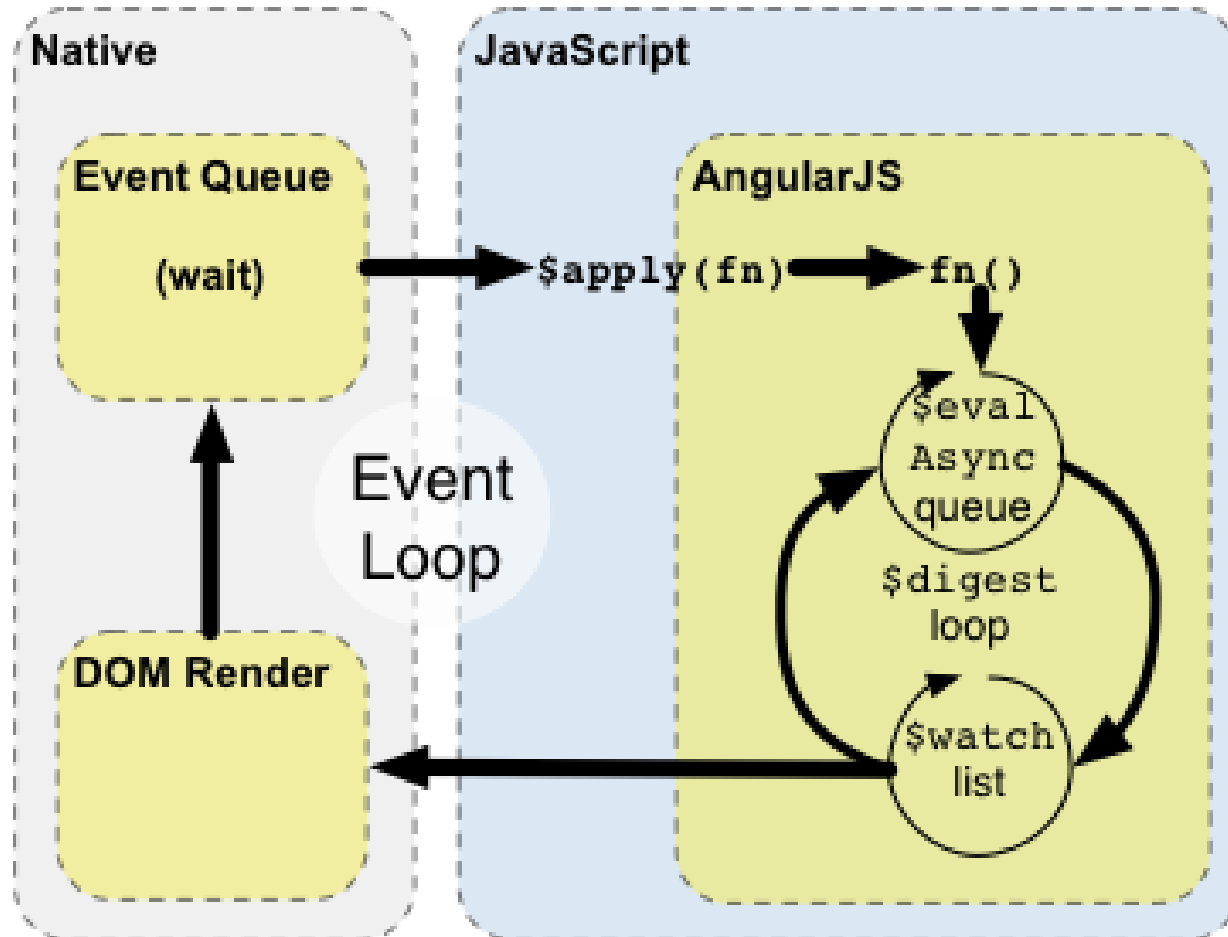
- Watching by reference ([scope.\\$watch](#))
 - **scope.\$watch** (watchExpression, listener)
- Watching collection contents ([scope.\\$watchCollection](#))
 - **scope.\$watchCollection** (watchExpression, listener)
- Watching by value
 - **scope.\$watch** (watchExpression, listener, **true**)

使用 \$watch 的注意事項 (監視深度)

```
$scope.users = [  
  {name: "Mary", points: 310},  
  {name: "June", points: 290},  
  {name: "Bob", points: 300}  
];
```



Angular 執行生命週期 (browser event loop)



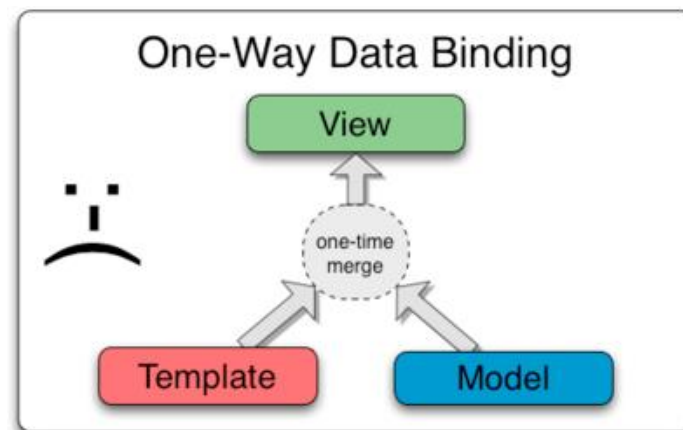
AngularJS 六大核心概念



1. 資料繫結 (Data-Binding)

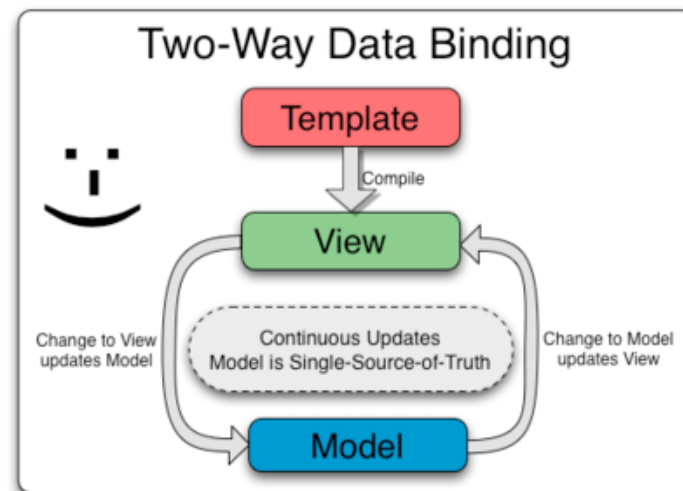
- 單向繫結 (One-way binding)

- 從 Model 繫結到 View
- 將資料自動呈現到網頁 DOM



- 雙向繫結 (Two-way binding)

- 從 View 繫結到 Model
- 透過「HTML 表單」輸入資料
- 頻繁檢查 (Dirty Check)



單次繫結 (One-time binding) (AngularJS 1.3+)

- 語法
 - `{{::name}}`
- 範例
 - <http://jsbin.com/fegabo/edit?html,js,output>

```
<div ng-controller="EventController">
  <button ng-click="clickMe($event)">Click Me</button>
  <p id="one-time-binding-example">One time binding: {{::name}}</p>
  <p id="normal-binding-example">Normal binding: {{name}}</p>
</div>
```

```
<ul>
  <li ng-repeat="item in ::items | orderBy:'name'">{{item.name}};</li>
</ul>
```

2. 宣告式編程 (Declarative Programming)

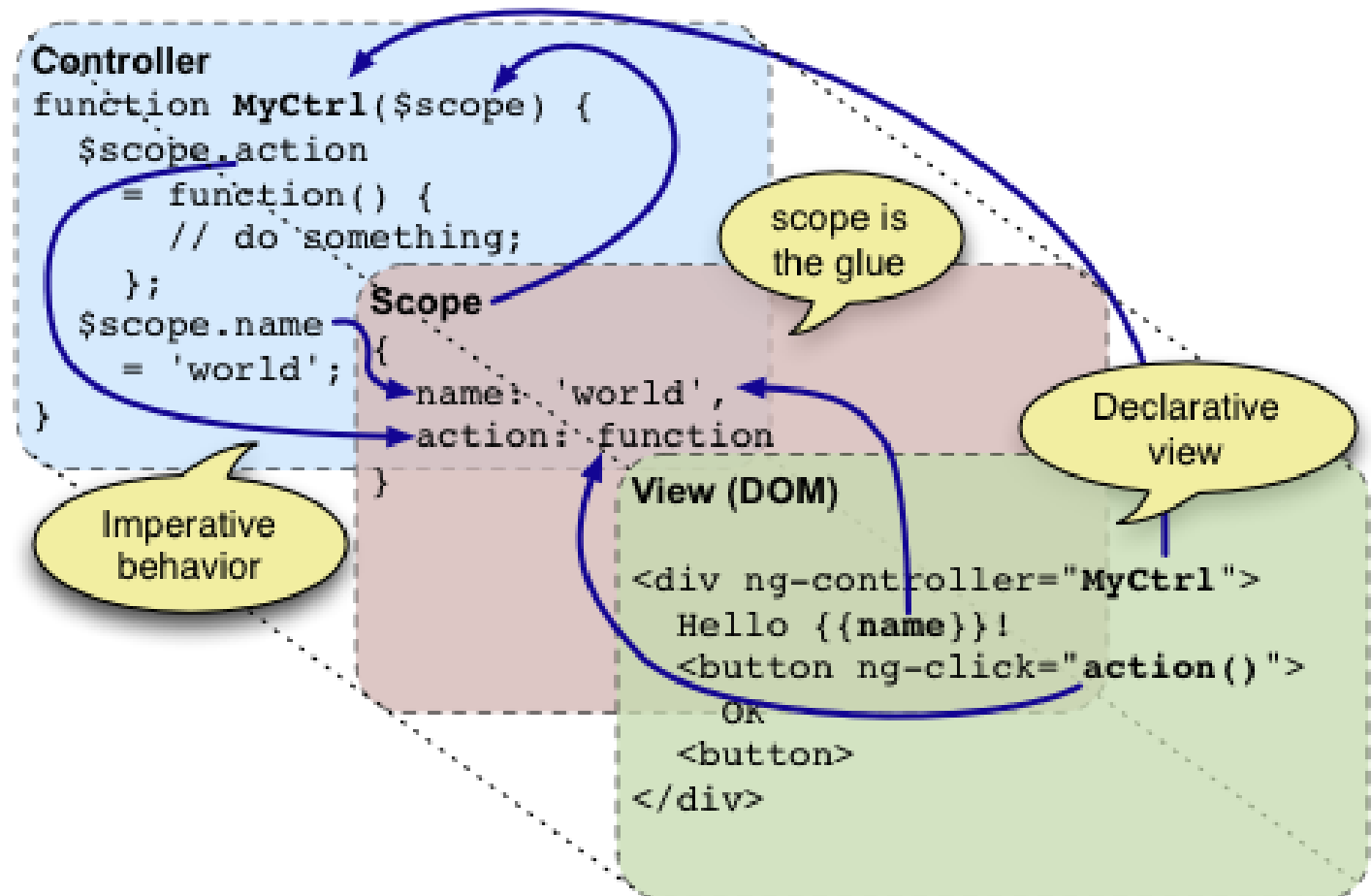
- 開發模式較為**抽象**，必須**抽象思考**
- 大部分動作透過「宣告」的方式即可直接使用
- 以 HTML 為主要範本 (Templates)
 - 屬性 (**A**tttributes)
 - 類別 (**C**lasses)
 - 元素 (**E**lements)
 - 註解 (**C**omments)

```
- <span my-dir="exp"></span>
  • <span ng:bind="name">
  • <span ng_bind="name">
  • <span ng-bind="name">
  • <span data-ng-bind="name">
  • <span x-ng-bind="name">
- <span class="my-dir: exp;"></span>
- <my-dir></my-dir>
- <!-- directive: my-dir exp -->
```

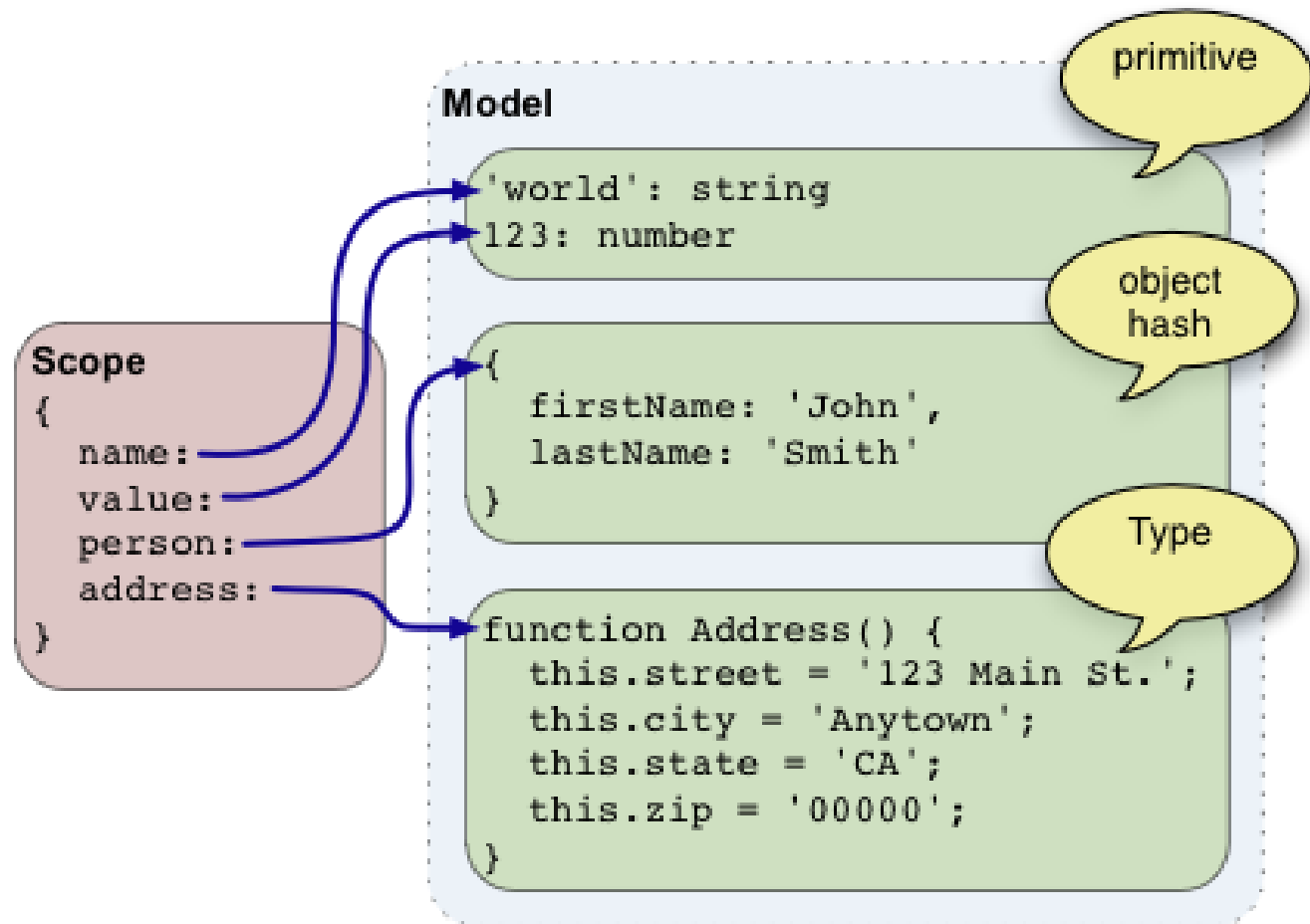

3. 關注點分離

- 透過模組化技術分離不同類型的工作 (MV*)
- 透過 MVC 設計樣式進行權責分工
 - Controller
 - 用來定義**應用程式**的主要行為! (商業邏輯)
 - 屬性 (原始型別、物件型別) / 事件 / 方法
 - Model
 - 用來存取資料，連結 View / Controller 的橋樑
 - `$scope` / `$rootScope` / `this`
 - View
 - 以 DOM 為範本 (相較於用 string 為範本)
 - 透過 ng-model 啟動**雙向**資料繫結
 - 原則: **不要把商業邏輯放在 View 裡面!**

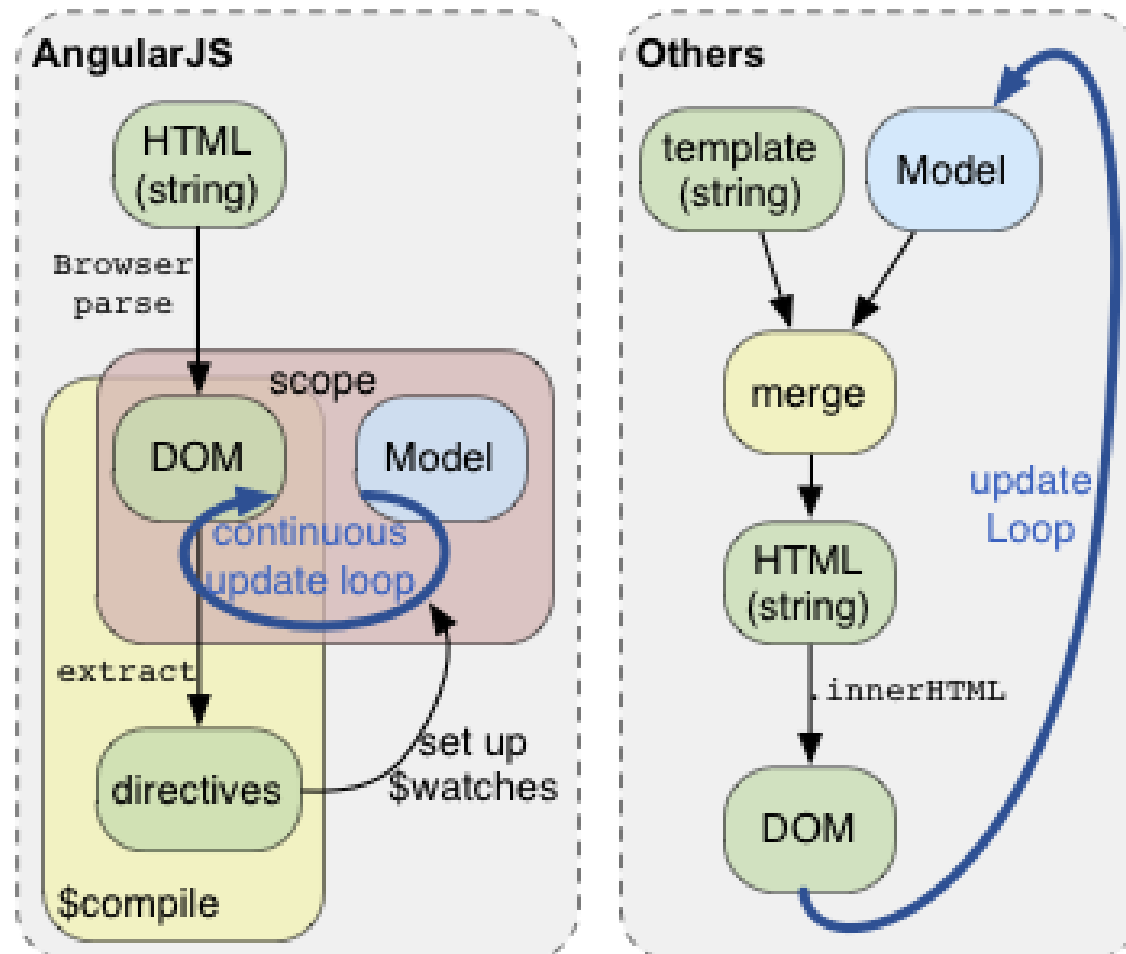
控制器 (Controller)



模型 (Model)



檢視頁面 (View)

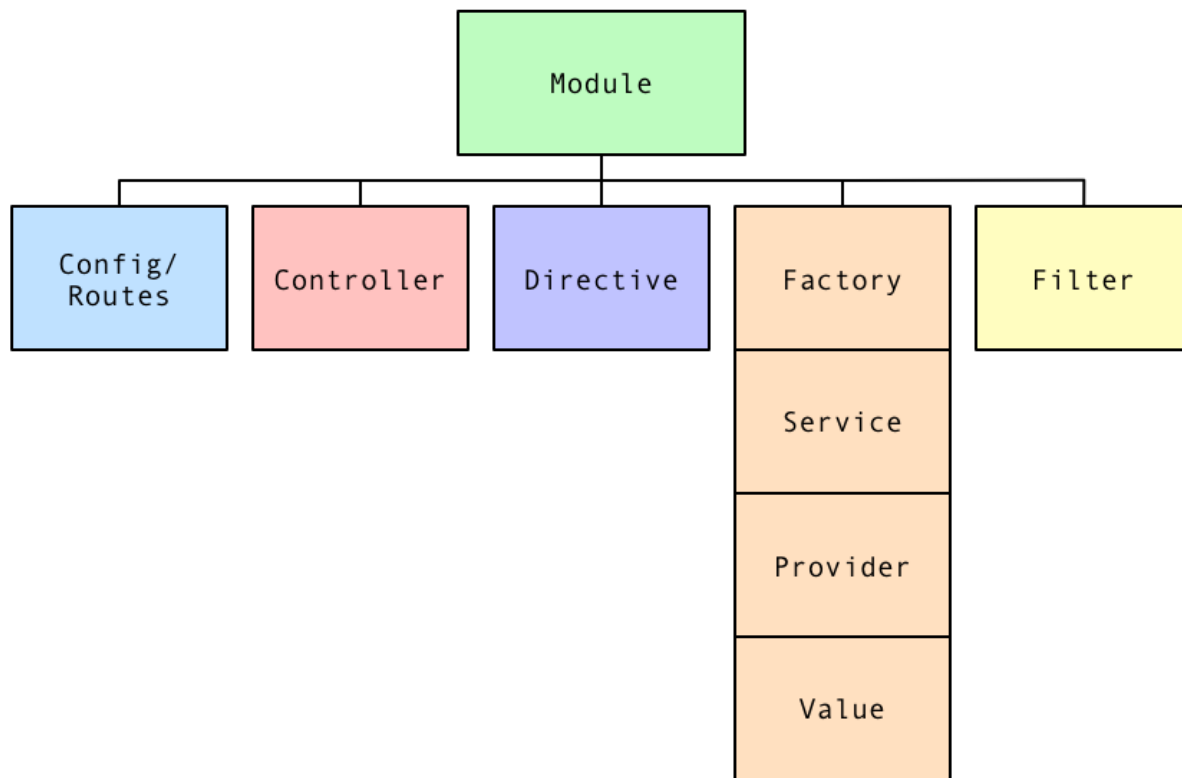


4. 相依性注入 (Dependency Injection)

- DI 是一種軟體設計樣式
 - 用來管理元件之間如何維護「相依性」
 - Angular 的 \$injector 用來負責：
 - 自動建立物件
 - 自動解析相依性
 - 提供其他元件取得建立後的物件
- AngularJS 模組化技術極度依賴 DI 機制
 - 相依性注入技術可大幅降低元件之間的耦合關係
 - 相依性注入技術可大幅降低元件設計的複雜度
 - 相依性注入技術可讓元件更容易實作單元測試

5. 擴充性

- 所有元件都可以自由擴充
 - modules
 - config
 - directives
 - controllers
 - filters
 - services
 - factory
 - service
 - provider
 - value
 - constant



6. 測試性

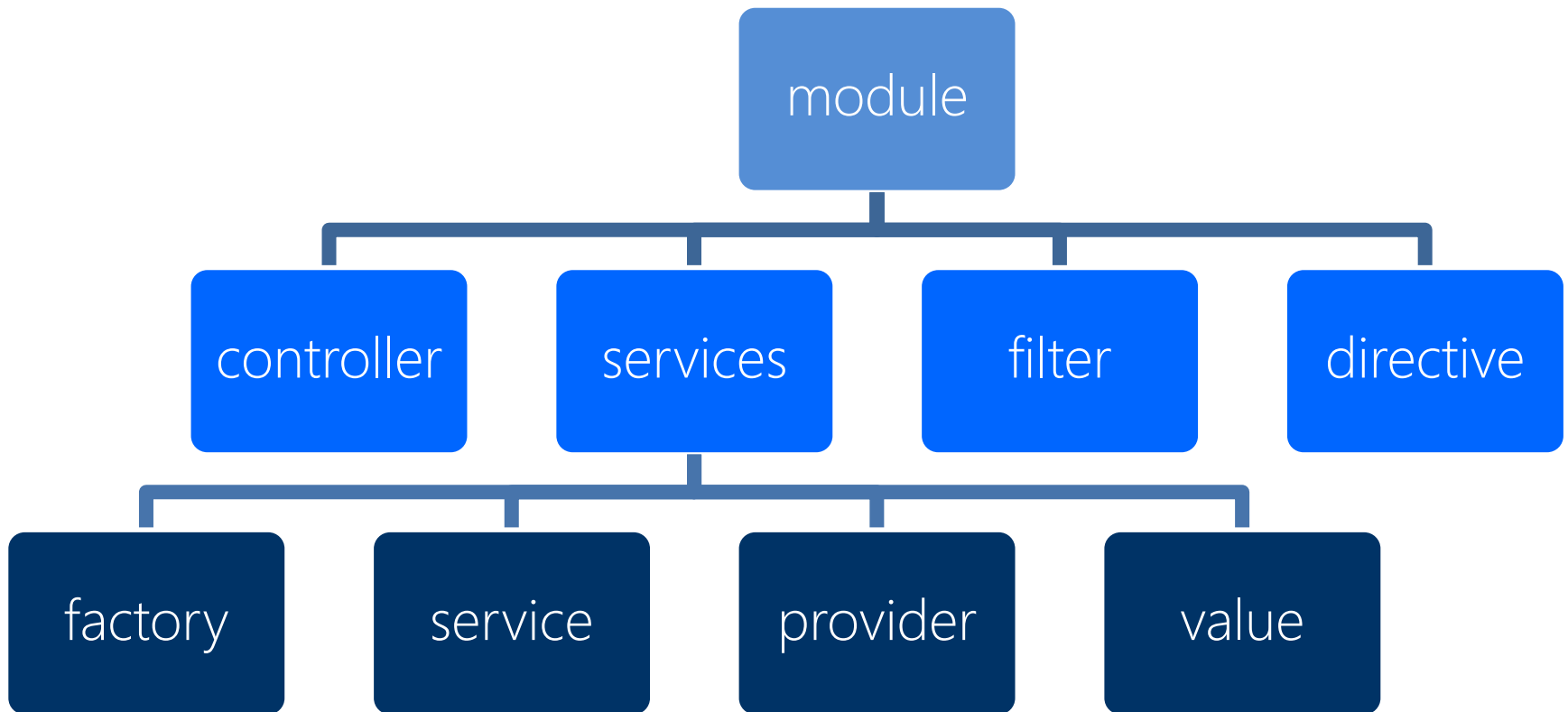
- 由於每個元件都可獨立存在，元件與元件之間透過 **DI** 機制管理相依性，因此每個元件都可以進行獨立測試，大幅降低測試難度
- 當一個元件用到其他元件時，即可透過 mock 的方式提供假元件注入，跳過相依元件的測試工作
- AngularJS 單元測試 (Unit Testing)
 - Karma - JavaScript Test Runner
 - Jasmine – BDD framework for JavaScript
- AngularJS 端對端測試 (E2E Testing)
 - Protractor - end to end test runner

Modularization

ANGULARJS 模組化技術



深入了解 Angular 模組 (Module)



深入了解 Angular 服務 (Services)

- 什麼是 Angular 服務？
 - 服務是一種可重複使用程式碼的概念
 - 可在整個應用程式中共用的行為(函式)與狀態(物件)
 - 以下元件都稱為服務元件
 - provider
 - service
 - factory
 - value
 - constant

常用 service 物件

- \$http
 - [http://docs.angularjs.org/api/ng.\\$http](http://docs.angularjs.org/api/ng.$http)
 - 範例:
 - moreText: 假文產生器
 - <http://jsbin.com/uyesej/2/edit>
 - <http://jsbin.com/uyesej/3/edit> (加上**更新**按鈕)
- \$log
 - [http://docs.angularjs.org/api/ng.\\$log](http://docs.angularjs.org/api/ng.$log)
- [\\$window](#) 與 [\\$document](#) 與 [\\$interval](#) 與 [\\$timeout](#)
 - window 與 window.document
 - angular.element(yourElement) == \$(yourElement)

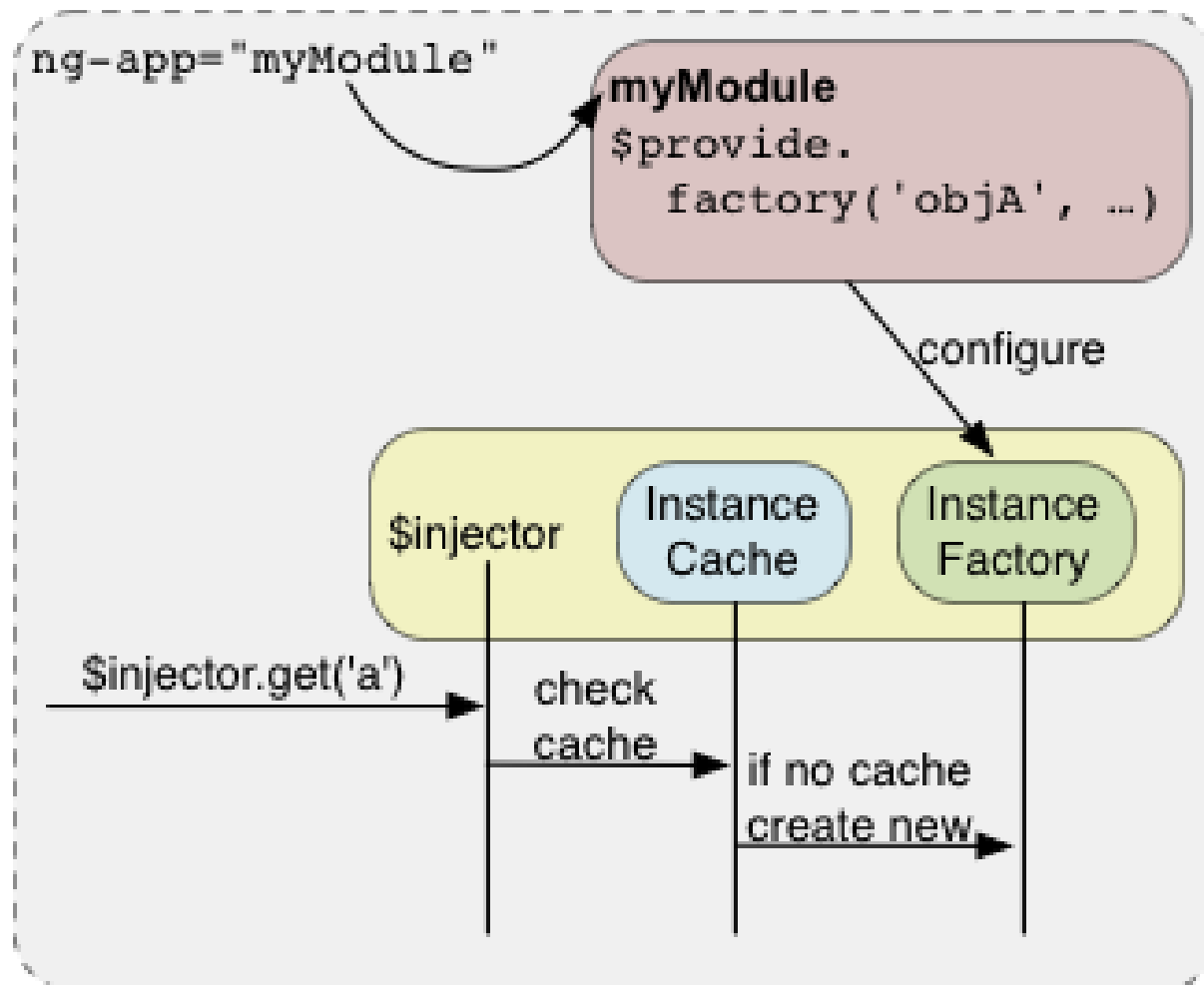
Angular 服務的特性

- 唯一的名稱
 - 每個服務都要有個唯一的名稱
- 獨體模式 (Singleton)
 - 每個同名的**服務**只會被 **實體化** 一次
 - 在整個應用程式裡，每次取得的**服務**都是相同物件實體
- 所有服務的背後都是個**提供者** (Provider)
 - 皆透過 [\\$provider](#) 註冊服務
 - 提供者名稱為 **服務名稱** + "**Provider**"
 - 例如：\$http 服務的提供者名稱就是 \$httpProvider
- 所有服務皆設計用來**注入**到其他**服務或控制器**中
 - 注入時皆由 [\\$injector](#) 進行存取

關於 Angular 提供者 (Provider)

- 運作原理
 - 由 [\\$provider](#) 註冊服務
 - 每個註冊的服務底層都有個 Provider 物件
 - 每個 Provider 物件都有個 **\$get** 方法
 - 這裡的 **\$get** 方法主要用來建立**服務實體**
- 提供者參數設定
 - 每個**提供者**在建立唯一的**服務實體**前，都可以進行設定
 - 你只能在 `myApp.config([ConfigFn])` 設定**提供者**
 - 大部分的服務都是不需要設定的，所以 Angular 提供好多種簡易的方法註冊服務，包括像是 `value`, `factory`, `service`, `constant` 這幾個 API

模組 (Modules) 與 注入器 (Injector)



提供者 (Provider) 的程式碼範例

```
.provider('MyProvider', [function () {  
    var counter = 0;  
    this.setCounter = function(i) {  
        counter = i;  
    };  
    this.$get = [function() {  
        return {  
            'getCounter': function() {  
                console.log('MyProvider: ' + counter++);  
            }  
        };  
    }];  
}]);  
}])
```

I

比較 Controllers 與 Services 的差別

- Controllers 的用途
 - 負責處理呈現邏輯 (Views)
 - 負責存取 Scope 資料 (Models)
 - 負責處理使用者觸發的事件 (Events)
- Controllers 的特性
 - 在一個應用程式生命週期中可被多次建立與摧毀
 - 具有短暫的狀態性
 - 兩個同階層的 Controller 不能互相交換資料
 - 雖然可以透過 \$parent Scope 交換資料，但這不是一個好解法！
- DEMO
 - 開啟範例【02 示範 Controller 的狀態性】
 - 執行 **webpack-dev-server** 並開啟 <http://localhost:8080>
 - 開啟 [ng-inspector](#) 擴充套件

比較 Controllers 與 Services 的差別

Controllers	Services
負責呈現邏輯 <ul style="list-style-type: none">• 取得資料• 處理互動• 控制 UI 顯示與樣式	負責工作邏輯 <ul style="list-style-type: none">• 與後端伺服器溝通• 資料驗證邏輯• 共用應用程式資料或狀態
與 view 相關	與 view 無關
特定目的使用或一次性呼叫	可不斷重用
可直接取用 Scope 物件	不可直接取用 Scope

Angular 服務的相依性注入

- 相依性注入的方法
 - 隱含相依性注入
 - 明確相依性注入 (避免 JS 最小化帶來的影響)

```
<script>
angular.module('myApp', [])
.controller('MainCtrl', ['$scope', '$http', function ($scope, $http) {
    var self = this;
    self.Name = "Will";

    $http.get('http://api.openweathermap.org/data/2.5/weather?q=Taipei,tw')
    .success(function(data, status, headers, config) {
        self.Weather = data;
    });

}]);
</script>
```

自訂 Angular 服務

- **設計準則 (心法)**

- 不要跟 view 有任何關係 (不要直接去處裡 DOM 物件)
- 放入可重複使用的**函式**或**物件** (因為**獨體模式**特性)
 - 保存**應用程式層級**的狀態
 - 需要被快取的物件 (不需要一直重新產生實體的物件)
- 需要與後端交換資料或與第三方服務整合時
- 不同的服務負責執行不同的任務，透過 **DI** 結合在一起

自訂 Angular 服務：value & constant

```
myApp.value('myvalue', {  
  'getCounter': function() {  
    console.log('MyFactory: ' + counter++);  
  }  
})
```

```
myApp.constant('constants', {  
  'APP_NAME': 'MyShoppingCart',  
  'APP_VERSION': '0.1',  
  'CartSaveUrl': '/api/carts/save'  
})
```

value 與 constant 的主要差異

- value
 - 無法注入到 `myApp.config()` 程式碼裡
 - 但可以注入到 controller 裡
- constant
 - 可以注入到 `myApp.config()` 與 controller 裡

自訂 Angular 服務：factory

```
myApp.factory('MyFactory', [function () {  
    var counter = 0;  
  
    return {  
        'getCounter': function() {  
            console.log('MyFactory: ' + counter++);  
        }  
    };  
}])
```

自訂 Angular 服務：service

```
myApp.service('MyService', [function () {  
    var counter = 0;  
  
    this.getCounter = function() {  
        console.log('MyService: ' + counter++);  
    };  
  
}])
```

自訂 Angular 服務：provider

```
myApp.provider('item', [function () {  
    var counter = 0;  
    this.setCounter = function(i) { counter = i; };  
    this.$get = [function() {  
        return {  
            'getCounter': function() {  
                console.log('MyProvider: ' + counter++);  
            }  
        };  
    }];  
}])
```


自訂 Angular 服務：provider (設定提供者)

```
myApp.config(['itemProvider',  
  function (itemProvider) {  
    itemProvider.setCounter(10);  
  }])
```

練習：自訂 Angular 服務

- 開啟【03 以 Angular 服務重構程式碼】專案
 - 自訂 **discount** 服務
 - 個別使用 service, factory, provider 建立服務
 - discount1 (service)
 - discount2 (factory)
 - discount3 (provider)
 - 修改 discount3 服務(Provider)
 - 讓 discount3 服務可在 module config 的地方設定折扣!
 - **\$discount3Provider.setDiscount(0.8);**

介紹服務修飾元件 (Service Decorators)

- Decorators 主要目的
 - 動態**新增**或**修改**內建服務或自訂服務的**行為**
 - 在不更動 Angular 服務程式碼的情況下進行**修飾**
- Decorators 主要特性
 - 實作 Decorators 設計樣式
 - 必須透過 \$provider 服務來宣告 Decorators

服務修飾元件 (Service Decorators) 範例

```
myApp.config('$provider', function($provider) {  
    $provider.decorator('$log', logDecorator);  
});
```

```
function logDecorator($original_log) {  
    return {  
        log: function(msg) {  
            $orig_log.log('-----');  
            $orig_log.log(msg);  
            $orig_log.log('-----');  
        };  
    };  
}
```

深入了解 Angular 過濾器 (Filter)

- 運作流程
 - 輸入資料
 - 處理資料
 - 輸出資料
- 特性
 - 宣告後預設會建立一個 `nameFilter` 服務(可注入)
- 程式碼範例

```
myApp.filter('item', function() {  
    return function(input) {  
        return input + "!";  
    };  
});
```

更多 Angular 過濾器 (Filter) 範例

```
myApp.filter('item', function() {  
  return function(input) {  
    return input + "!";  
  };  
});
```

```
myApp.controller('MainController', function($filter) {  
  var itemFilter = $filter('item');  
  itemFilter(input_obj);  
});
```

```
myApp.controller('MainController', function(itemFilter) {  
  itemFilter(input_obj);  
});
```

<https://github.com/a8m/angular-filter>

Angular 過濾器 (Filter) 注意事項

- 使用考量
 - 注意 Angular 的頻繁檢查 (dirty check) 特性
 - 在 view 使用 filter 效率極差，盡量在 controller/service 中使用，因為 filter 會在每次 digest 迴圈中被執行
 - 搭配 ng-repeat 使用時，更要特別小心 (還是因為 digest 迴圈)
- 設計考量
 - filter 的執行時間不能過長
 - 避免 DOM 操作、呼叫外部資源、非同步工作、長時間工作
 - 不要在 filter 中觸發額外的 digest 迴圈

深入了解 Angular 指令 (Directive)

- 什麼是 Angular 指令？
 - Directive 是一種將 **呈現** (HTML) 與 **行為** (JS) 封裝在一起的 UI 元件，可讓 HTML 變得**更加抽象化**、**更具有語意**，並且更容易**重複使用**，讓使用者只須要知道元件的運作方式，不需要知道繁瑣的內部細節！
 - Directive 主要目的用來**控制 DOM 物件**的變化
 - Directive 次要目的則是用來**修飾網頁的預設行為**
- 自訂 Directive 的替代方案
 - ng-include
 - ng-switch

建立自訂的 Angular 指令 (Directive)

```
myApp.  
  .directive('name', [function () {  
    return {  
      restrict: 'A',  
      templateUrl: 'name-template.html',  
      controller: 'A',  
      scope: {},  
      link: function ($scope, $element, $attrs) {  
  
      }  
    };  
  }])
```

Angular 指令 (Directive) 的命名規則

- Directive 命名
 - 預設使用 camelCase (駝峰式大小寫) 來命名
 - `directive('cartSummary', [directiveFn])`
- 屬性名稱
 - `<div cart-summary>`
- 標籤名稱
 - `<cart-summary></cart-summary>`
- 類別名稱
 - `<div class="cart-summary">`
- 註解語法
 - `<!-- directive: cart-summary -->`

載入動態網頁範本或靜態網頁範本

- 動態網頁範本：templateURL (從網址載入範本)
- 靜態網頁範本：template (用字串定義範本)

myApp.

```
.directive('cartSummary', [function () {  
  return {  
    restrict: 'AE',  
    templateUrl: 'cart-summary-tp1.html',  
  };  
}])
```

Angular 指令 (Directive) 的使用限制

- restrict
 - 屬性 (**A**tttributes)
 - 類別 (**C**lasses)
 - 元素 (**E**lements)
 - 註解 (**C**omments)
- 可自由組合使用，例如：
 - restrict: 'EA'

Angular 指令 (Directive) 的 link 函式

- link 函式
 - 如 Directive 的「建構式」函式
 - 當 Directive **被建立時**會自動執行的程式碼 (僅執行一次)
 - 有點類似 View (HTML) 所屬的 Controller 程式碼
- 使用方法

```
link: function ($scope, $element, $attrs) {  
}
```
- 參數說明
 - `$scope` 該 Directive 當下的範圍物件
 - `$element` 該 Directive 的 jqLite 元素物件
 - `$attrs` HTML 屬性值

Angular 指令 (Directive) 的 compile 函式

- compile 函式
 - 如 Directive 正在被編譯(解析)的時候會執行 compile 函式(僅一次)
 - 此函式是操作 DOM 物件的最佳時機
 - 該函式必須回傳一個 link 函式
 - 注意：compile 與 link 絕對不會一起使用！
- 使用方法

```
compile: function compile(tElement, tAttrs) {  
    return function postLink(scope, iElement, iAttrs, controller) {  
    }  
}
```

- 參數說明
 - tElement 該 Directive 範本的 jqLite 物件
 - tAttrs 該 Directive 範本的 HTML 屬性值

Angular 指令 (Directive) 的範圍定義

- 範圍定義 (scope)
 - 預設 Directive 會自動沿用他的上層範圍物件且自動傳入 link 函式中，因此該 Directive 與所屬範圍的 Directive 將會共用完全相同的 \$scope 物件。
- 使用方法
 - false 預設值，代表沿用他的上層範圍物件
 - true 繼承上層範圍物件（會自行建立子範圍物件）
 - {} 不繼承上層範圍物件（建立隔離範圍）
 - = 雙向繫結
 - @ 單向繫結
 - & 函式繫結

各種 scope 參數使用範例

```
myApp.  
  .directive('cartSummary', [function () {  
    return {  
      restrict: 'AE',  
      scope: true,  
      templateUrl: 'cart-summary-tp1.html',  
    };  
  }])
```

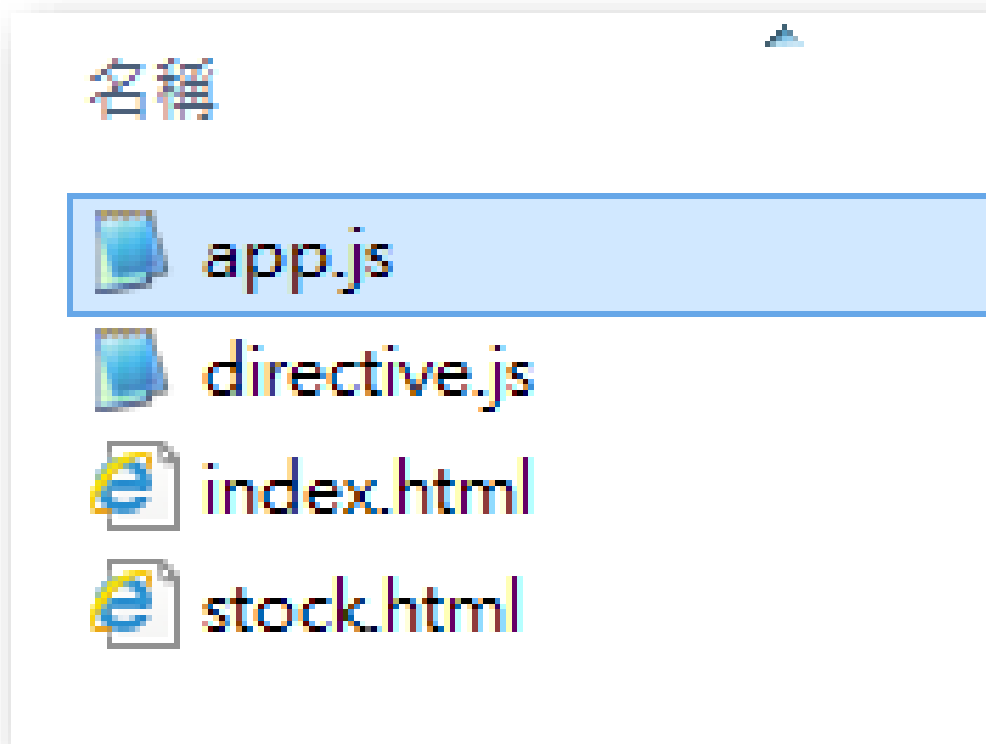

Angular 指令 (Directive) 的範圍定義

- 隔離物件的進階用法

```
scope: {  
    // 將 HTML 的 my-model 屬性繫結到上層範圍物件  
    // 在 link 函式中，可以取得 $scope.myModel 的資料 (且雙向繫結)  
    myModel: '=',  
  
    // 將 HTML 的 my-title 屬性繫結到字串，該字串會自動解析表達式  
    // 在 link 函式中，可以取得 $scope.myTitle 的資料 (且單向繫結)  
    myTitle: '@'  
  
    // 將 HTML 的 my-action 屬性繫結到上層範圍物件的特定函式  
    // 在 link 函式中，可以執行 $scope.myAction 並傳入一個物件參數  
    // 傳入參數(物件)的屬性名稱，就是在使用 directive 時的參數名稱  
    myAction: '&'  
}
```

各種 scope 參數使用範例

- 04 Directives 範例程式\directive-with-scope-advanced



Angular 指令 (Directive) 的取代行為

- `replace: false`
 - 預設值為 `false`
- `replace: true`
 - 使用該 `directive` 的標籤將會被完整取代為 `directive` 的 `template` 內容

※ 注意：AngularJS 1.3 之後不建議再使用

Advanced Topics

ANGULARJS 進階議題



介紹 AngularJS 好用開發工具

- 開發環境
- 開發工具/編輯器
- 測試與偵錯工具

建立 AngularJS 開發環境

- [RubyInstaller](#) / [Compass](#)
 - 安裝 Ruby 1.9.3-p551
 - `gem update --system`
 - `gem install compass`
- [LiveReload](#)
 - [LiveReload 0.9.2 Alpha](#) (Win) / [瀏覽器擴充套件](#) (Safari, Chrome, Firefox)
 - `npm install -g livereload`
 - `start livereload .`
- [webpack](#)
 - `npm install -g webpack-dev-server webpack`
 - `webpack-dev-server`
- [yeoman](#)
 - `npm install -g bower gulp yo generator-angular-webpack`
 - `yo angular-webpack`
 - `gulp`

選擇 AngularJS 開發工具/編輯器

- [Sublime Text 3](#)
 - [AngularJS](#) (官方套件好用)
 - [Using the AngularJS Package for Sublime Text](#)
 - [AngularJS Snippets](#) (內建許多 Snippets 超好用)
- [WebStorm](#)
 - [Using AngularJS](#)
 - [AngularJS Workflow in WebStorm](#)
- [Visual Studio 2013](#) (請升級到 Update 4 更新)
 - [AngularJS SPA Template](#)
 - AngularJS SPA Template (SPA 架構漂亮，Web API 需額外安裝)
 - [SideWaffle Template Pack](#)
 - AngularJS And Web API – Empty (架構乾淨，但無範例程式)
 - [TypeScript Tools for Microsoft Visual Studio 2013 1.4.0.0](#)

選擇 AngularJS 偵錯工具

- 偵錯工具
 - Chrome 擴充套件
 - [ng-inspector for AngularJS](http://ng-inspector.org/) (<http://ng-inspector.org/>)
 - [Angular Scope Inspector](#)
 - [AngularJS Batarang \(Stable\)](#) v0.4.3 [建議安裝穩定版]
 - [AngularJS Batarang](#) v0.8.1 [還有 [Bug](#) 正在修]
 - Firefox 擴充套件
 - [ng-inspector for AngularJS](#)
 - [AngScope](#)
 - 書籤列工具
 - [ng-stats](#)
 - GitHub: <https://github.com/kentcdodds/ng-stats>

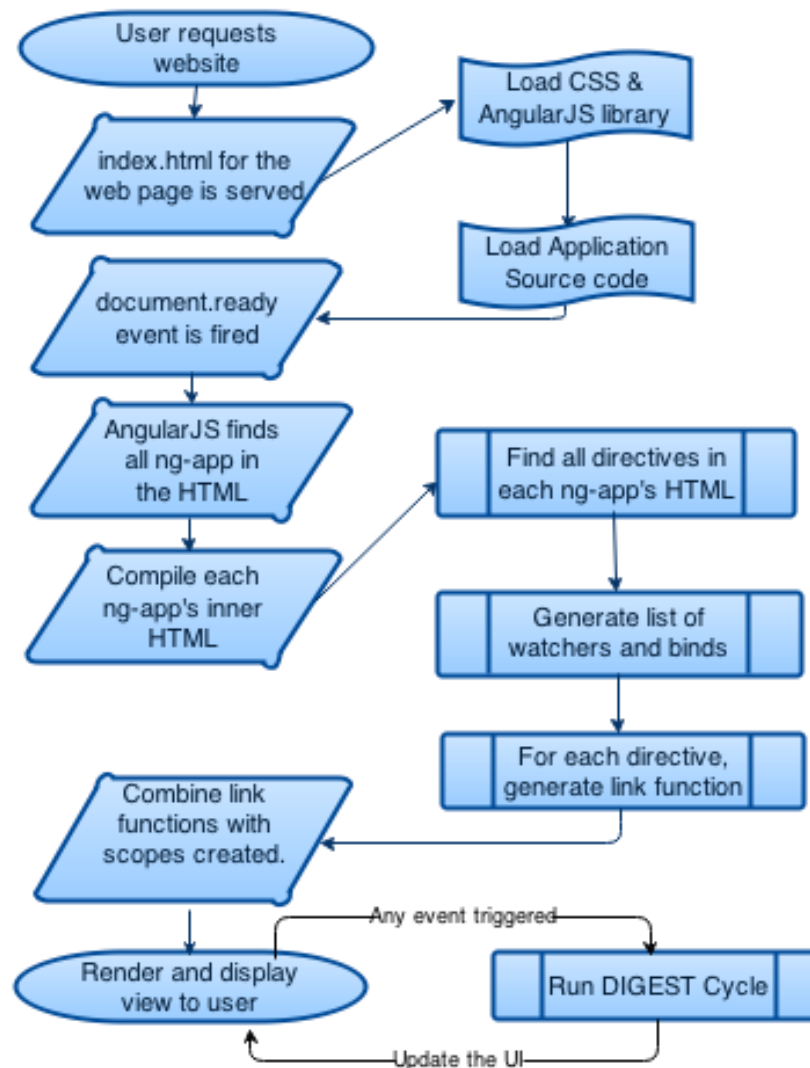
選擇 AngularJS 測試工具

- 測試工具
 - [Karma](#)
 - [Jasmine](#)
 - [MochaJS](#)
 - [Protractor](#)
- 效能測試
 - [Benchpress](#)

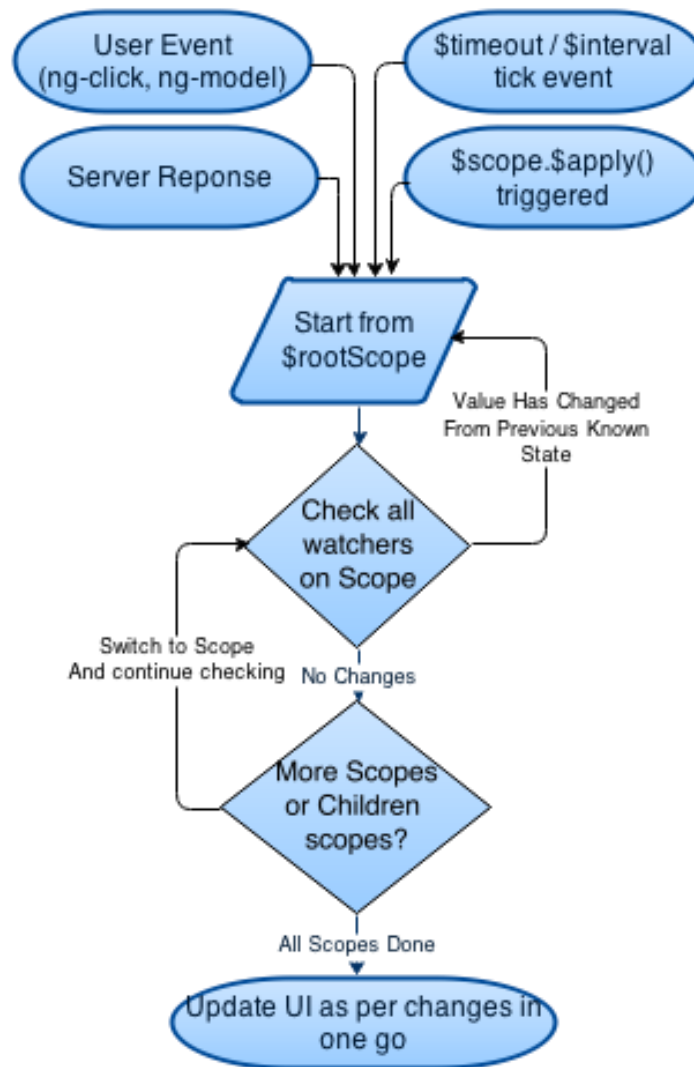
了解 AngularJS 效能調校方法

- 先搞清楚不同元件負責不同工作
 - Controllers
 - 執行所有與 View 相關的邏輯 (不含 DOM 操作)
 - Services
 - 執行所有可獨立完成且重複的工作
 - Directives
 - 執行所有與 DOM 相關的工作 (包含與外部函式庫整合)
 - Filters
 - 執行所有資料格式化與資料過濾的工作
- 再搞清楚應用程式與各元件的「生命週期」細節
 - Digest Loop
 - \$watchers

AngularJS 的初始生命週期



AngularJS 的 digest 生命週期



AngularJS 的 directive 生命週期

- 在 `.directive('stockWidget', [function() {...}])` 函式 return 之前執行的 code 只會執行一次
- 第一次使用 directive 的時候會載入範本，並存入 `$templateCache` 之中，之後便不會再更新
- directive 的範本被編譯 (`$compile`) 並且產生一個 link 函式
- directive 的 Scope 物件被建立
 - parent scope
 - child scope of the parent scope
 - isolated scope
- directive 的 link 函式會在每次建立 directive 實體時被執行

AngularJS 效能調校文章

- [AngularJS Performance Tips](#)
- [Speeding up AngularJS apps with simple optimizations](#)
- [11 Tips to Improve AngularJS Performance](#)
- [AngularJS Performance Tuning for Long Lists](#)
- [Optimizing AngularJS: 1200ms to 35ms](#)
- [Angular Performance Tips](#)

AngularJS 效能調校的基本原則

- 減少 Watches 數量
- 減少 Digest Loop 的迴圈次數
- 多用 ng-bind 少用 {{ }}，善用 One-time binding
- 盡量不要再 Views 裡面使用 filter



Learning Resources for AngularJS

ANGULARJS 學習資源

保哥分享的相關學習資源

- 文章分享
 - The Will Will Web | AngularJS
<http://blog.miniasp.com/category/AngularJS.aspx>
- 影片分享
 - AngularJS 開發實戰：重要的開發觀念與經驗分享 (COSCUP2013)
<http://www.youtube.com/watch?v=aXuK2ACHLcU>
 - AngularJS 的 ngSwitch 指令 (directive) 使用陷阱與範圍觀念
<http://www.youtube.com/watch?v=l8cqOJp6xyw>
- 簡報分享
 - AngularJS 開發實戰：解析 angular-seed 專案架構與內容
<http://www.slideshare.net/WillHuangTW/angularjs-angularseed>
 - JavaScript 開發實戰：效能調校與常見陷阱 (2013 JSDC.tw)
<http://www.slideshare.net/WillHuangTW/java-script-jsdc2013>
 - JavaScript 物件導向觀念入門 v.s. TypeScript 開發實戰
<http://www.slideshare.net/WillHuangTW/type-script-20528669>

AngularJS 相關學習資源

- [AngularJS Hub](#) (從大量的範例中學習 AngularJS 開發)
- [AngularJS-Learning](#)
- [Learn AngularJS from the best](#)
- [Egghead.io – AngularJS](#) (免費教學影片) (YouTube)
- [Egghead.io – AngularJS](#) (付費教學影片)
- [AngularJS Cheat Sheet by ProLoser](#)
- [Pluralsight - Search results for: angular](#)
- [Mastering AngularJS | WintellectNOW](#)
- [25 days of AngularJS Calendar | 2013](#)
- [Angular Style Guide](#)

聯絡資訊

- The Will Will Web

記載著 Will 在網路世界的學習心得與技術分享

- <http://blog.miniasp.com/>

- Will 保哥的技術交流中心 (臉書粉絲專頁)

- <http://www.facebook.com/will.fans>



- Will 保哥的噗浪

- <http://www.plurk.com/willh/invite>

- Will 保哥的推特

- https://twitter.com/Will_Huang