**CÂU HỎI PHỎNG VẤN REACT + ANGULAR**

**Mục Lục**

[1. Chia sẻ dữ liệu giữa các component 4 cách : 1](#_Toc517737931)

[2. Phân biệt @ViewChild @ViewChildren vs @ContentChild @ContentChildren 2](#_Toc517737932)

[3. LifecycleHook Angular : 2](#_Toc517737933)

[4. Service trong Angular : 3](#_Toc517737934)

[5. So Sánh Angular và ReactJS : 3](#_Toc517737935)

[6. Pure Function : 4](#_Toc517737936)

[7. React Lifecycle Hook : 4](#_Toc517737937)

[8. Mutable và Immutable : 5](#_Toc517737938)

[9. HTML5 vs HTML : 6](#_Toc517737939)

[10. CSS3 6](#_Toc517737940)

# 1. Chia sẻ dữ liệu giữa các component 4 cách :

1. Input, Output + Event Emitter
2. Con => Cha : Viewchild.

( Tại cha gọi **@ViewChild(ChildComponent) child** để truy xuất các thuộc tính và phương thức tại component con)

1. Con => Cha : ContentChildrent

( Tại cha gọi **@ContentChildren(ChildComponent) child: QueryList<ChildComponent>** )

\_ Ta được danh sách các component con. Ta có thể filter và truy xuất từng thuộc tính và phương thức con

1. Service : cả 2 component sẽ inject service.
2. Routing.
3. Ngrx : **~Redux**

\_ Lưu vào store sau đó có thể gọi ở truy cập ở component bất kỳ.

# 2. Phân biệt @ViewChild @ViewChildren vs @ContentChild @ContentChildren

\_ Một ứng dụng Angular được tạo thành từ một số thành phần lồng nhau theo 2 cách : cách thông thường và cách dùng ng-content. Để truy cập thành phần con ta dùng các decorator trên.

\_ Child và Childrent khác nhau là kiểu dữ liệu trả về là số nhiều và duy nhất 1 .

\_ So sánh **view** và **content** :

+ Giống nhau : cùng thực hiện 1 chức năng là tham chiếu đến một cái gì đó trong template ( html ). Trong js thuần : document.getElementById('some-id'); angular sinh ra là tránh làm việc trực tiếp với DOM. Sử dụng @ViewChildren và ContentChildren cả 2 sẽ trả về một QueryList cung cấp một quan sát ( observable ) mà bạn có thể theo dõi các thay đổi. Tốt hơn so với việc dùng thuần.

+ Khác nhau :

|  |  |
| --- | --- |
| View | Content |
| **không** bao gồm các phần tử tồn tại trong thẻ ng-content. | bao gồm các phần tử tồn tại trong thẻ ng-content. |
| ViewChildren chỉ được tạo trong lifecycle AfterViewInit được chạy. | ContentChildren được tạo trong vòng đời AfterContentInit được chạy. |
| @ViewChild và @ViewChildren chỉ có thể lấy trực tiếp các phần tử trong view, dùng quan hệ cha con cách thông thường. | Các truy vấn @ContentChild và @ContentChildren sẽ trả về các chỉ thị tồn tại bên trong phần tử <ng-content> </ ng-content> . Dùng quan hệ cha con bằng ng-content. |

# 3. LifecycleHook Angular :

\_ Dùng chung cho cả Directive và Component :

* **ngOnInit:** Khởi tạo directive và component sau khi Angular khởi tạo các thuộc tính của dữ liệu ràng buộc
* **ngOnChanges:** Phản hồi lại sau khi angular cài đặt một thuộc tính dữ liệu ràng buộc. Phương thức nhận một object changes của hiện tại và giá trị trước đó, gọi trước **ngOnInit** 1 lần duy nhất
* **ngDoCheck:** Phát hiện và thực hiện thay đổi. Được gọi mỗi khi tìm thấy thay đổi
* **ngOnDestroy:** Thu hồi trước khi Angular hủy bỏ directive/component. Hủy đăng ký observables và tháo gỡ các sự kiện để tránh tốn bộ nhớ

**\_** Chỉ Component :

* **ngAfterContentInit**: Sau khi angular xuất content ra view
* **ngAfterContentChecked**: Sau khi Angular kiểm tra sự ràng buộc content để xuất content ra view
* **ngAfterViewInit**: Sau khi Angular tạo ra view của componet
* **ngAfterViewChecked**: Sau khi Angular kiểm tra ràng buộc dữ liệu trên view

\_ Lưu ý :

* Constructor là phương thức khởi tạo mặc định của class, thực hiện khi class được khởi tạo.
* NgOnInit là một hook vòng đời, được gọi bởi Angular2 để chỉ rõ Angular đã hoàn thành việc tạo ra component.
* Không nên lấy dữ liệu về khi khởi tạo bởi constructor theo khuyến cáo lập trình thì càng gọn nhẹ càng tốt. Vì vậy, nên sử dụng NgOnInit để lấy dữ liệu vào component ngay sau khi khởi tạo

# 4. Service trong Angular :

\_ Là những hệ thống con có những nhiệm vụ khác nhau : logging, truy cập dữ liệu, caching, ...

\_ Là một class ES6 đóng gói một vài chức năng và cung cấp nó cho phần còn lại của ứng dụng.

\_ Để sử dụng : khai báo provider cho module hoặc riêng component, sau đó tiêm vào constructor và sử dụng.

# 5. So Sánh Angular và ReactJS :

\_ Cả 2 đều dùng module đóng gói webpack.

\_ Kiến trúc dựa theo component

|  |  |
| --- | --- |
| **Angular** | **ReactJS** |
| Là một web framework đầy đủ. | Là một thư viện, đi cùng với các thư viện redux, flux, mobx. |
| Cơ chế phát hiện thay đổi. | Cập nhật DOM nhanh dựa vào DOM ảo. |
| DOM thật | DOM ảo |
| Render tại client và server | Render tại server |
| Two way binding : thực hiện đồng bộ dữ liệu từ Component vs DOM và ngược lại | One way binding |
| Quản lý denpendency bằng công cụ được xây dựng sẵn để quản lý sự phụ thuộc ( service ) | Cần ReactDI |
| Typescript, JS | JS, JSX |
| Tuân theo quy ước | Dễ custom |
| Cần kiến thức JS => ES6 => TS | JS |
| Typechecking trong angular : các biến và hàm sẽ được khai báo mở loại dữ liệu | Trong react có PropTypes để check kiểu dữ liệu các props nhận vào. |
| Google | Facebook |
| Create React App | Angular CLI |

# 6. Pure Function :

\_ Là hàm luôn trả về kết quả giống nhau khi truyền vào tham số giống nhau.

\_ Không phụ thuộc vào các biến bên ngoài.

VD :

var tax = 10;

function calculateTax(productPrice) { return (productPrice \* (tax/100)) + productPrice; }

=> Hàm này phụ thuộc vào biến tax bên ngoài hàm.

# 7. React Lifecycle Hook :

**ComponentWillMount**

Khi chúng ta gọi một Component thì hàm này sẽ được gọi trước khi hàm render được thực thi

**componentDidMount**

Hàm này chỉ thực hiện 1 lần duy nhất.Thực hiện sau khi render lần đầu tiên bên phía client. Đây là thời điểm lý tưởng để gọi các AJAX trên cũng như các xử lý với DOM.

**componentWillReceiveProps**

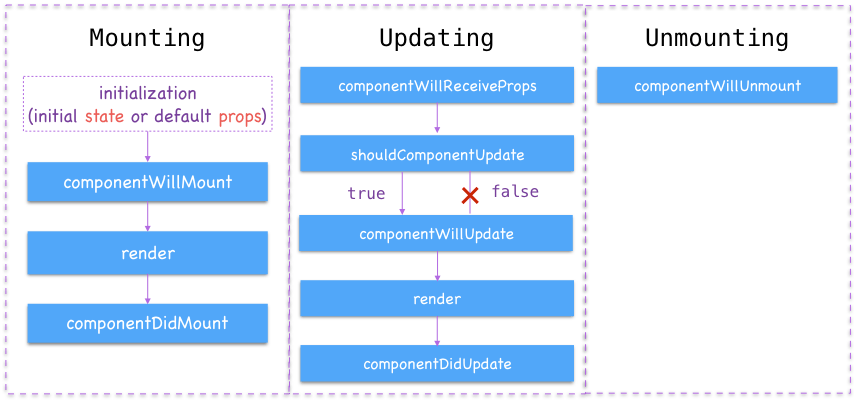
Hàm này được thực thi liên tục ngay khi các Props có sự thay đổi và trước khi một render khác được gọi. Thường sử dụng để thay đổi trạng thái (state) của component phụ thuộc props

**shouldComponentUpdate**

Hàm này trẻ về kết quả True or False.Dùng để xác định Component có update hay không.Mặc định giá trị trả về là True.nếu bạn muốn Component không cần render sau khi Props hoặc Statet hay đổi thì bạn có thể set giá trị trả về là False.

**componentWillUnmount**

Hàm này được gọi sau khi component unmounted từ DOM



# 8. Mutable và Immutable :

\_ Mutable : giá trị thay đổi được. Kiểu Date trong java. Tự do thay đổi cấu trúc dữ liệu ban đầu => thay đổi các đối tượng liên quan đến nó => khó debug.

\_ Immutable : giá trị không thể thay đổi được. Tránh các bug liên quan đến reference. Giúp dev tăng khả năng dự đoàn biến, debug, cải thiện performance.

Khi gán giá trị vào 1 immutable object là chúng ta sẽ không thay đổi giá trị mà là tạo ra 1 instance mới. Sau đó sao chép các thuộc tính hiện có + giá trị mới thêm vào. Tất cả bản update đều trả về giá trị mới.

Lợi ích : dễ dàng xem sự thay đổi. VD về thay đổi state của react component. Ta có thể dùng shouldComponentUpdate để kiểm tra xem trạng thái hiện tại và trước đó có giống nhau không để không cần phải re-render lại.

\_ Phương thức : Immutable.fromJS, get, first, last, sort, sortBy, groupBy…

# 9. HTML5 vs HTML :

\_ Các thẻ mới seo tốt : header, nav, section, article, aside, footer.

\_ Lưu trữ ngoại tuyến.

\_ Cho phép tương tác với hình ảnh, biểu đồ, lập trình game dùng canvas.

\_ Định vị người dùng ( geolocation ).

\_ Hỗ trợ API

\_ Web storage trong html5 : an toàn hơn cookie, nhanh, dữ liệu lớn : sessionStorage, localStorage.

# 10. CSS3

\_ Animation

\_ Multibackground

\_ Empty : dùng :empty

\_ Resize

\_ Pseudo class : nth(2)

\_ TextShadow