

ChatGPT

Bạn đóng vai trò là một mentor chuyên nghiệp để dạy tôi Angular từ cơ bản đến nâng cao và luôn luôn diễn giải chi tiết cặn kẽ và code đầy đủ các bước để chạy được ví dụ. Bạn đã dạy tôi các bài học sau đây: Bài 1: Giới thiệu Angular và Cài đặt Môi trường Bài 2: Data Binding trong Angular Bài 3: Component Communication trong Angular Bài 4: Angular Routing (Điều hướng giữa các Component) Bài 5: Reactive Forms trong Angular Bài 6: Gọi API trong Angular với HttpClient Hãy tiếp tục với Bài 7: Quản lý trạng thái trong Angular với NgRx (State Management)

Bài 7: Quản lý trạng thái trong Angular với NgRx (State Management)

1. Giới thiệu về Quản lý trạng thái trong Angular

Trong một ứng dụng Angular, việc quản lý trạng thái (state management) rất quan trọng, đặc biệt là khi ứng dụng trở nên phức tạp với nhiều component chia sẻ dữ liệu. **NgRx** là một thư viện mạnh mẽ giúp quản lý trạng thái dựa trên mô hình **Redux**.

Lợi ích của NgRx

- **Đồng nhất dữ liệu:** Trạng thái của ứng dụng được lưu trữ trong một Store duy nhất.
- **Tối ưu hiệu suất:** Giảm thiểu việc truyền dữ liệu giữa các component, hạn chế re-render không cần thiết.

- **Dễ dàng debug:** NgRx có DevTools hỗ trợ theo dõi lịch sử thay đổi của trạng thái.
- **Quản lý trạng thái theo mô hình unidirectional data flow (luồng dữ liệu một chiều)** giúp dễ hiểu và bảo trì hơn.

2. Cấu trúc của NgRx trong Angular

NgRx hoạt động theo mô hình **Redux** với 4 thành phần chính:

1. **Actions (Hành động):** Định nghĩa các sự kiện xảy ra trong ứng dụng.
2. **Reducers (Bộ giảm):** Xử lý sự kiện và cập nhật trạng thái mới.
3. **Store (Kho lưu trữ):** Lưu trữ trạng thái ứng dụng và cung cấp nó cho các component.
4. **Selectors (Bộ chọn):** Truy vấn trạng thái từ Store.

Dưới đây là sơ đồ tổng quan về cách hoạt động của NgRx:

```
mathematica
```

```
Component → Dispatch Action → Reducer → Store → Component
```

3. Cài đặt NgRx vào dự án Angular

Trước khi sử dụng NgRx, bạn cần cài đặt các package sau:

```
sh
```

```
ng add @ngrx/store
ng add @ngrx/effects
ng add @ngrx/store-devtools
ng add @ngrx/entity
ng add @ngrx/router-store
```

Sau khi cài đặt, bạn có thể bắt đầu sử dụng NgRx để quản lý trạng thái.

4. Ví dụ: Quản lý danh sách sản phẩm bằng NgRx

Chúng ta sẽ tạo một ứng dụng Angular sử dụng NgRx để quản lý danh sách sản phẩm.

Bước 1: Tạo một dự án mới (Nếu chưa có)

```
sh
```

```
ng new nxrx-demo --style=scss --routing
cd nxrx-demo
```

Bước 2: Tạo một thư mục quản lý trạng thái

Trong thư mục `src/app`, tạo thư mục `store/products/` và các file như sau:

```
markdown
```

```
store/
├── products/
│   ├── product.model.ts
│   ├── product.actions.ts
│   ├── product.reducer.ts
│   ├── product.selectors.ts
│   ├── product.effects.ts
│   └── product.service.ts
```

Bước 3: Định nghĩa Model (product.model.ts)

typescript

```
export interface Product {  
  id: number;  
  name: string;  
  price: number;  
}
```

Bước 4: Định nghĩa Actions (product.actions.ts)

Actions là các sự kiện mà ứng dụng có thể thực hiện, như **tải danh sách sản phẩm, thêm sản phẩm mới, xóa sản phẩm**.

typescript

```
import { createAction, props } from '@ngrx/store';  
import { Product } from '../product.model';  
  
export const loadProducts = createAction('[Product] Load Products');  
export const loadProductsSuccess = createAction(  
  '[Product] Load Products Success',  
  props<{ products: Product[] }>()  
)  
);  
export const loadProductsFailure = createAction(  
  '[Product] Load Products Failure',  
  props<{ error: string }>()  
)  
);
```

Bước 5: Tạo Reducer để xử lý trạng thái (product.reducer.ts)

Reducer nhận vào trạng thái hiện tại và action, sau đó trả về trạng thái mới.

typescript

```
import { createReducer, on } from '@ngrx/store';
import { Product } from '../product.model';
import * as ProductActions from '../product.actions';

export interface ProductState {
  products: Product[];
  loading: boolean;
  error: string | null;
}

export const initialState: ProductState = {
  products: [],
  loading: false,
  error: null
};

export const productReducer = createReducer(
  initialState,
  on(ProductActions.loadProducts, state => ({
    ...state,
    loading: true
  })),
  on(ProductActions.loadProductsSuccess, (state, { products }) => ({
    ...state,
    loading: false,
    products
  })),
  on(ProductActions.loadProductsFailure, (state, { error }) => ({
    ...state,
    loading: false,
    error
  })))
);
```

Bước 6: Tạo Selectors để truy vấn trạng thái (product.selectors.ts)

Selectors giúp component lấy dữ liệu từ Store một cách dễ dàng.

typescript

```
import { createSelector, createFeatureSelector } from '@ngrx/store';
import { ProductState } from '../product.reducer';

export const selectProductState = createFeatureSelector<ProductState>
('products');

export const selectAllProducts = createSelector(
  selectProductState,
  (state) => state.products
);

export const selectLoading = createSelector(
  selectProductState,
  (state) => state.loading
);

export const selectError = createSelector(
  selectProductState,
  (state) => state.error
);
```

Bước 7: Sử dụng Store trong Component

Thêm Store vào module (app.module.ts)

typescript

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { StoreModule } from '@ngrx/store';
import { productReducer } from './store/products/product.reducer';

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    StoreModule.forRoot({ products: productReducer })
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Sử dụng Store trong Component (app.component.ts)

typescript

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
import { Product } from '../store/products/product.model';
import { loadProducts } from '../store/products/product.actions';
import { selectAllProducts, selectLoading } from
'../store/products/product.selectors';

@Component({
  selector: 'app-root',
  templateUrl: '../app.component.html',
  styleUrls: ['../app.component.scss']
})
export class AppComponent {
  products$: Observable<Product[]>;
  loading$: Observable<boolean>;

  constructor(private store: Store) {
    this.products$ = this.store.select(selectAllProducts);
    this.loading$ = this.store.select(selectLoading);
  }

  loadProducts() {
    this.store.dispatch(loadProducts());
  }
}
```

Giao diện hiển thị sản phẩm (app.component.html)

html

```
<button (click)="loadProducts()">Load Products</button>

<div *ngIf="loading$ | async">Loading...</div>

<ul>
  <li *ngFor="let product of products$ | async">
    {{ product.name }} - {{ product.price | currency }}
  </li>
</ul>
```

5. Kết luận

- **NgRx giúp quản lý trạng thái một cách hiệu quả** trong các ứng dụng Angular lớn.
- **Sử dụng Actions để gửi sự kiện**, Reducers để cập nhật trạng thái, và Selectors để lấy dữ liệu từ Store.
- **Store giúp component lấy dữ liệu mà không cần truyền trực tiếp qua Input/Output**, làm code dễ bảo trì hơn.

Bạn có muốn mở rộng ứng dụng này bằng cách thêm API hoặc hiệu ứng (Effects) để lấy dữ liệu không? 🚀

Đổi sang ví dụ về counter cho dễ hình dung hơn

Bài 7: Quản lý trạng thái trong Angular với NgRx (State Management) - Ví dụ Counter

1. Giới thiệu

Trong bài này, chúng ta sẽ xây dựng một **Counter App** (Ứng dụng đếm số) sử dụng **NgRx** để quản lý trạng thái. Đây là một ví dụ đơn giản nhưng giúp bạn hiểu cách hoạt động của **Actions, Reducers, Store, và Selectors** trong NgRx.

2. Cài đặt NgRx vào dự án

Trước tiên, nếu bạn chưa có dự án Angular, hãy tạo một dự án mới:

```
sh
```

```
ng new ngx-counter --style=scss --routing  
cd ngx-counter
```

Tiếp theo, cài đặt các thư viện cần thiết của NgRx:

```
sh
```

```
ng add @ngrx/store  
ng add @ngrx/store-devtools
```

3. Cấu trúc thư mục

Tạo một thư mục **store/counter/** trong **src/app/** để quản lý trạng thái Counter:

```
markdown
```

```
store/  
├─ counter/  
│   ├── counter.actions.ts  
│   ├── counter.reducer.ts  
│   └── counter.selectors.ts
```

4. Tạo Actions (counter.actions.ts)

Actions giúp định nghĩa các sự kiện mà ứng dụng có thể thực hiện, như tăng, giảm hoặc reset giá trị counter.

typescript

```
import { createAction } from '@ngrx/store';

// Định nghĩa các action
export const increment = createAction('[Counter] Increment');
export const decrement = createAction('[Counter] Decrement');
export const reset = createAction('[Counter] Reset');
```

5. Tạo Reducer (counter.reducer.ts)

Reducer nhận trạng thái hiện tại và Action, sau đó trả về trạng thái mới.

typescript

```
import { createReducer, on } from '@ngrx/store';
import * as CounterActions from './counter.actions';

// Trạng thái ban đầu của counter
export const initialState = 0;

// Định nghĩa reducer
export const counterReducer = createReducer(
  initialState,
  on(CounterActions.increment, (state) => state + 1),
  on(CounterActions.decrement, (state) => state - 1),
  on(CounterActions.reset, () => 0)
);
```

6. Tạo Selectors (counter.selectors.ts)

Selectors giúp lấy trạng thái từ Store để sử dụng trong component.

typescript

```
import { createSelector, createFeatureSelector } from '@ngrx/store';

// Lấy trạng thái counter từ Store
export const selectCounter = createFeatureSelector<number>('counter');
```

7. Thêm Store vào App Module (app.module.ts)

Bây giờ chúng ta cần đăng ký Store vào module chính của ứng dụng.

typescript

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { StoreModule } from '@ngrx/store';
import { StoreDevtoolsModule } from '@ngrx/store-devtools';
import { counterReducer } from '../store/counter/counter.reducer';
import { AppComponent } from '../app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    StoreModule.forRoot({ counter: counterReducer }),
    StoreDevtoolsModule.instrument({ maxAge: 25 }) // Hỗ trợ debug với Redux DevTools
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

8. Sử dụng Store trong Component (app.component.ts)

Bây giờ chúng ta sẽ sử dụng Store để hiển thị giá trị **counter** và xử lý các sự kiện tăng, giảm, reset.

typescript

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
import { increment, decrement, reset } from
'./store/counter/counter.actions';
import { selectCounter } from './store/counter/counter.selectors';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  counter$: Observable<number>;

  constructor(private store: Store) {
    this.counter$ = this.store.select(selectCounter);
  }

  increase() {
    this.store.dispatch(increment());
  }

  decrease() {
    this.store.dispatch(decrement());
  }

  reset() {
    this.store.dispatch(reset());
  }
}
```

9. Giao diện hiển thị Counter (app.component.html)

html

```
<div class="container">
  <h1>Counter: {{ counter$ | async }}</h1>
  <button (click)="increase()">+</button>
  <button (click)="decrease()">-</button>
  <button (click)="reset()">Reset</button>
</div>
```

10. CSS (app.component.scss)

scss

```
.container {
  text-align: center;
  margin-top: 50px;

  h1 {
    font-size: 3rem;
    margin-bottom: 20px;
  }

  button {
    font-size: 1.5rem;
    margin: 5px;
    padding: 10px 20px;
    cursor: pointer;
  }
}
```

11. Kết quả

Khi chạy ứng dụng bằng lệnh:

sh

ng serve

Bạn sẽ thấy giao diện hiển thị số counter, và khi nhấn các nút, trạng thái sẽ được cập nhật thông qua **NgRx**.

12. Tóm tắt

- **Actions** giúp định nghĩa các sự kiện như **increment**, **decrement**, **reset**.
 - **Reducer** cập nhật trạng thái dựa trên các Actions.
 - **Store** lưu trữ trạng thái của ứng dụng.
 - **Selectors** giúp truy vấn trạng thái từ Store.
 - **Component** lắng nghe trạng thái từ Store và dispatch Actions để thay đổi trạng thái.
-



Mở rộng

- Thêm **NgRx Effects** để lưu trạng thái vào **localStorage**.
- Hiển thị **lịch sử thay đổi Counter** bằng Redux DevTools.
- Kết hợp với **Angular Material** để làm giao diện đẹp hơn.

Bạn có muốn mở rộng thêm tính năng nào không? 😊

ChatGPT can make mistakes. Check important info.