**生命周期**

**1 - ROOT\_EFFECTS\_INIT**

添加了所有root effects之后，root effects分派一个ROOT\_EFFECTS\_INITaction。您可以将此action看作一个生命周期钩子，在添加了所有root effects之后，您可以使用它来执行一些代码。

init$ = createEffect(() =>

this.actions$.pipe(

ofType(ROOT\_EFFECTS\_INIT),

map(action => ...)

)

);

**2 - Effect Metadata**

Non-dispatching Effects

有时，您不希望effects分派action，例如，当您只想基于传入action进行日志记录或导航时。但是当一个effect没有分派另一个action时，浏览器就会崩溃，因为这个effect同时“订阅”和“分派”了完全相同的动作，从而导致一个无限循环。为了防止这种情况发生，可以在createEffect函数中添加{dispatch: false}作为第二个参数。

@Injectable()

export class LogEffects {

constructor(private actions$: Actions) {}

logActions$ = createEffect( () =>

this.actions$.pipe( tap(action => console.log(action)) ),

{ dispatch: false });

}

**3 -Resubscribe on Error**

从版本8开始，当一个错误在effects的主流中发生时，它会用Angular的ErrorHandler来报告，并且源effects会被自动重新订阅(而不是完成)，所以它会继续监听所有分派的动作。默认情况下，effects被重新订阅多达10个错误。

一般来说，错误应该由用户处理。但是，对于遗漏错误的情况，这个新行为增加了一个额外的安全网。

在某些使用特定RxJS操作符的情况下，新的行为可能会产生意想不到的结果。例如，如果startWith操作符位于效果的管道中，那么它将再次被触发。

要禁用重新订阅，请向createEffect元数据(第二个参数)添加{useEffectsErrorHandler: false}。

@Injectable()

export class AuthEffects {

logins$ = createEffect(

() =>

this.actions$.pipe(

ofType(LoginPageActions.login),

exhaustMap(action =>

this.authService.login(action.credentials).pipe(

map(user => AuthApiActions.loginSuccess({ user })),

catchError(error => of(AuthApiActions.loginFailure({ error })))

)

)

// 错误得到处理，并且禁用重新订阅是安全的 ),

{ useEffectsErrorHandler: false }

);

constructor( private actions$: Actions, private authService: AuthService ) {}

}

**4 - Customizing the Effects Error Handler**

通过使用EFFECTS\_ERROR\_HANDLER注入令牌提供自定义处理程序，可以定制默认的重新订阅处理程序的行为。这允许您提供自定义行为，例如仅重试某些“可重试”错误，或更改重试的最大次数(默认设置为10次)。

import { EffectsModule, EFFECTS\_ERROR\_HANDLER } from '@ngrx/effects';

import { MovieEffects } from './effects/movie.effects';

import { CustomErrorHandler, isRetryable } from '../custom-error-handler';

export function effectResubscriptionHandler>T extends Action<(

observable$: Observable>T<, errorHandler?: CustomErrorHandler): Observable>T< {

return observable$.pipe(

retryWhen(errors =>

errors.pipe(

mergeMap(e => {

if (isRetryable(e)) {

return errorHandler.handleRetryableError(e);

}

errorHandler.handleError(e);

return throwError(e);

}) ) ) );}

@NgModule({

imports: [EffectsModule.forRoot([MovieEffects])],

providers: [

{

provide: EFFECTS\_ERROR\_HANDLER,

useValue: effectResubscriptionHandler,

},

{

provide: ErrorHandler,

useClass: CustomErrorHandler

} ],})

**控制作用 (Controlling Effects)**

**1 - OnInitEffects**

实现此接口以在添加效果后分派自定义操作。您可以在应用程序的其余部分中侦听此操作，以便在效果注册后执行某些操作。

class UserEffects implements OnInitEffects {

ngrxOnInitEffects(): Action {

return { type: '[UserEffects]: Init' };

}

}

**2 -OnRunEffects**

默认情况下，效果被合并并订阅到存储区。实现OnRunEffects接口来控制已解析效果的生命周期。

@Injectable()

export class UserEffects implements OnRunEffects {

constructor(private actions$: Actions) {}

updateUser$ = createEffect(() =>

this.actions$.pipe(

ofType('UPDATE\_USER'),

tap(action => {

console.log(action);

})

),

{ dispatch: false });

ngrxOnRunEffects(resolvedEffects$: Observable<EffectNotification>) {

return this.actions$.pipe(

ofType('LOGGED\_IN'),

exhaustMap(() =>

resolvedEffects$.pipe(

takeUntil(this.actions$.pipe(ofType('LOGGED\_OUT')))

) ) ); }}

**3 -Identify Effects Uniquely**

默认情况下，无论加载Effect类多少次，每个Effect类都注册一次。通过实现此接口，您可以定义一个惟一标识符来多次注册Effect类实例。

class EffectWithIdentifier implements OnIdentifyEffects {

constructor(private effectIdentifier: string) {}

ngrxOnIdentifyEffects() {

return this.effectIdentifier;

}

}

**4 - Effects operators**

作为Effects库的一部分，NgRx提供了一些常用的有用操作符。

ofType

ofType操作符根据字符串值(表示操作类型)或操作创建者过滤操作流。

必须提供动作<TypeUnion>的泛型，以便对字符串值进行类型推断。基于createAction函数的Action creator没有同样的局限性。

ofType操作符最多使用5个参数，并具有适当的类型推断。它可以采取更多，但是类型将被推断为一个Action接口。

@Injectable()

export class AuthEffects {

login$ = createEffect(() =>

this.actions$.pipe(

// Filters by Action Creator 'login'

ofType(LoginPageActions.login),

exhaustMap(action =>

this.authService.login(action.credentials).pipe(

map(user => AuthApiActions.loginSuccess({ user })),

catchError(error => of(AuthApiActions.loginFailure({ error })))

)

)

)

);

constructor(

private actions$: Actions,

private authService: AuthService

) {}

}