



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE

# Angular Testing



## 4 - RxJs Marbles

# RxJs Marbles

- Special Notation
- Primarily made for internal usage
- Use Cases:
  - Complex Operator and multiple values
  - Custom Operators
- No support for asynchronicity outside of operators
  - Promises
  - `setTimeout`, `setInterval`



# Marble Diagram

`const observable = m.cold(`  or 

`'--a-b-c';` sequence of elements  
(1 frame = 1 virtual millisecond)

`{ a: 2, b: 10, c: 25 }`

`);` values of elements



# Testing Structure

```
import { marbles } from 'rxjs-marbles/jest';
import { map } from 'rxjs/operators';

test(
  'default check',
  marbles((m) => {
    const source$ = m.cold('--a-b-c', { a: 2, b: 10, c: 25 });

    const destination$ = source$.pipe(map((n) => n * 2));

    m.expect(destination$).toBeObservable(
      '--x-y-z', { x: 4, y: 20, z: 50, });
  })
);
```



# Advanced Features

- Synchronous Events

- `'abcd(e|)'`

gets element and completes

- Setting Time

- `m.cold('d 2ms p 2ms h')`

time between elements



# Advanced Features - Flushing

```
let searchCounter = 0;

const source = m.cold('abcdef', {...});

const destination = source.pipe(
  tap(() => searchCounter++),
);

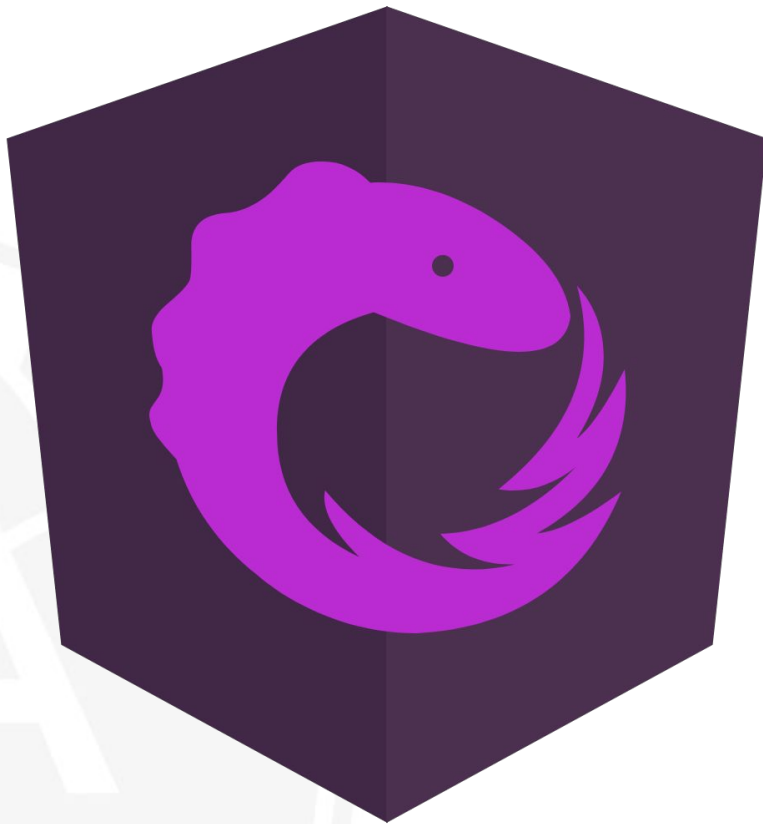
m.expect(destination).toBeObservable('abcdef', {...});

m.flush();

expect(searchCounter).toBe(6);
```



# Testing NgRx



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE

# Reducer 1/2

```
export const holidaysReducer = createReducer<HolidaysState>(
  initialState,
  on(holidaysActions.findHolidaysSuccess, (state, { holidays }) => ({
    ...state,
    holidays
  })))
);
```





# Reducer 2/2

```
it('should add the holidays on findHolidaySuccess', () => {  
  const holidays = [  
    { id: 1, title: 'Pyramids' },  
    { id: 2, title: 'Tower Bridge' }  
  ] as Holiday[];  
  
  const state = holidaysReducer(  
    { holidays: [] },  
    holidaysActions.findHolidaysSuccess({ holidays })  
  );  
  
  expect(state).toEqual({ holidays });  
});
```



# Selectors 1/2

```
const stateSelector = createFeatureSelector<HolidaysState>(holidaysFeatureKey);
```

```
export const fromHolidays = {  
  get: createSelector(stateSelector, ({ holidays }) => holidays)  
};
```



# Selectors 2/2

```
it('should select the holidays', () => {  
  const state: HolidaysState = {  
    holidays: [  
      { id: 1, title: 'Pyramids' },  
      { id: 2, title: 'Tower Bridge' }  
    ] as Holiday[]  
  };  
  expect(fromHolidays.get.projector(state)).toEqual([  
    { id: 1, title: 'Pyramids' },  
    { id: 2, title: 'Tower Bridge' }  
  ]);  
});
```



# Effects 1/2

```
export class HolidaysEffects {  
  find$ = createEffect(() =>  
    this.actions$.pipe(  
      ofType(holidaysActions.findHolidays),  
      switchMap(() =>  
        this.httpClient.get<{ holidays: Holiday[] }>(   
          'https://eternal-app.s3.eu-central-1.amazonaws.com/holidays.json'  
        )  
      ),  
      map(({ holidays }) => holidaysActions.findHolidaysSuccess({ holidays })))  
    );  
  constructor(private actions$: Actions, private httpClient: HttpClient) {}  
}
```



# Effects 2/2

```
it(  
  'should test find$',  
  marbles((m) => {  
    const httpClient = {  
      get: () => m.cold('---a', { a: { holidays: [{ id: 1 }] } })  
    };  
    const actions$ = m.cold('a', { a: holidaysActions.findHolidays() });  
  
    const effect = new HolidaysEffects(actions$, (httpClient as unknown) as HttpClient);  
  
    m.expect(effect.find$).toBeObservable('---a', {  
      a: holidaysActions.findHolidaysSuccess({ holidays: [{ id: 1 } as unknown] as Holiday })  
    });  
  })  
);
```



Lab Time

