Caching Observables

Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata


Alarm
Net
HDD
IR
Esc
Fn
Enter
100-240VAC

Caching Observables

Retain retrieved data locally

Reuse previously retrieved data

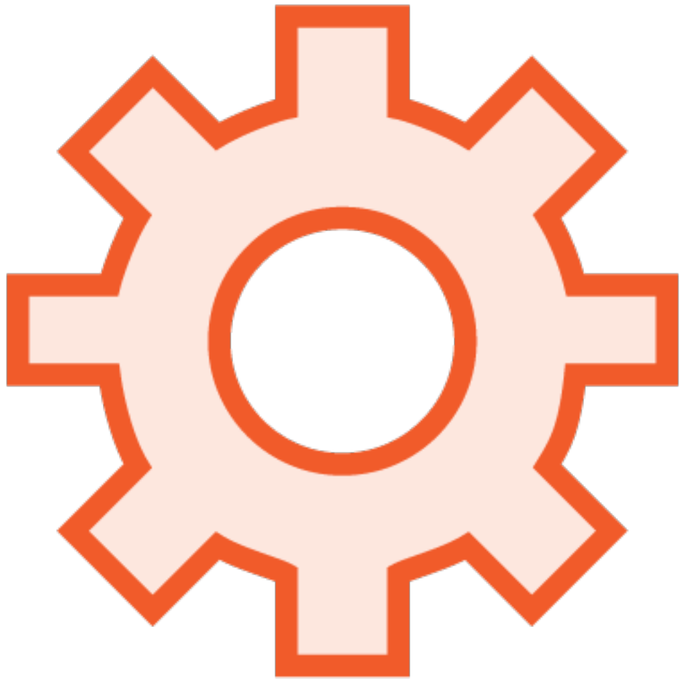
Stored in memory or external

Module Overview

Why caching?

Patterns for data caching

RxJS Features



`shareReplay`

`share`

Retrieving Data

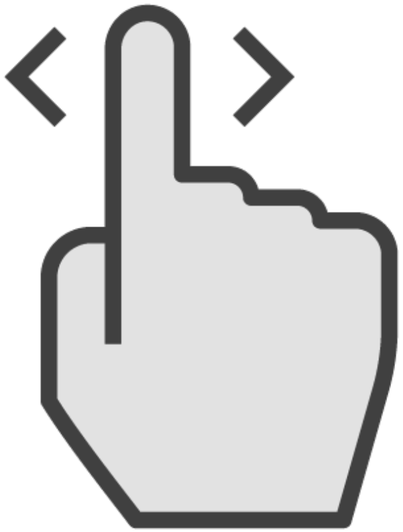
Product List Component

```
products: Product[] = [];

constructor(private productService: ProductService) { }

ngOnInit(): void {
  this.productService.getProducts()
      .subscribe(products => this.products = products);
}
```

Product List Template

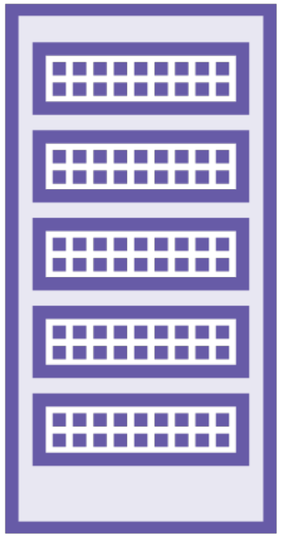
```
<div *ngIf="products$ | async as products">
<table>
  <tr *ngFor="let product of products">
    <td>{{ product.productName }}</td>
    <td>{{ product.productCode }}</td>
  </tr>
</table>
```

Advantages of Caching Data

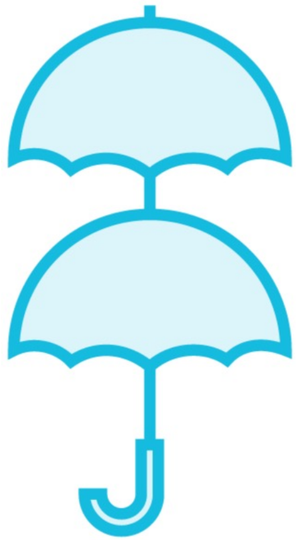


Improves responsiveness

Reduces bandwidth and network consumption



Reduces backend server load



Reduces redundant computations

Classic Caching Pattern

Product Service

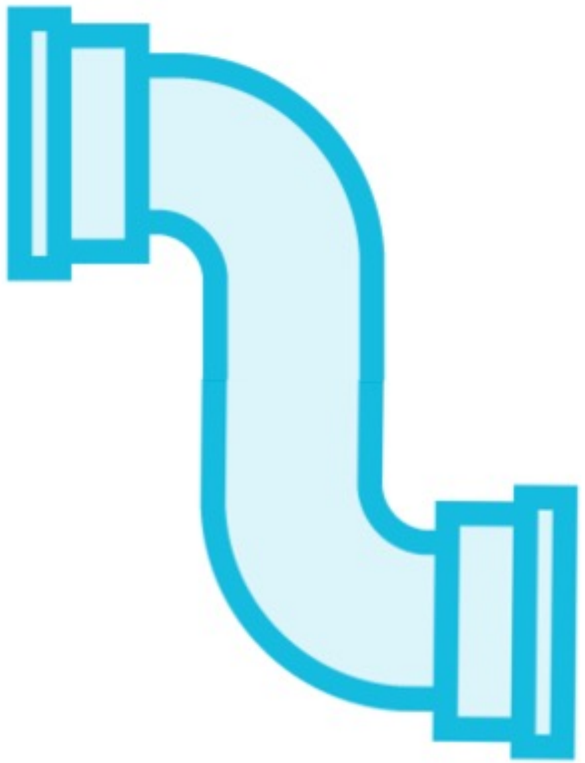
```
private products: Product[] = [];

getProducts(): Observable<Product[]> {
  if (this.products) {
    return of(this.products);
  }
  return this.http.get<Product[]>(this.url)
    .pipe(
      tap(data => this.products = data),
      catchError(this.handleError)
    );
}
```

Declarative Caching Pattern

Product Service

```
private url = 'api/products';

products$ = this.http.get<Product[]>(this.url)
  .pipe(
    shareReplay(1),
    catchError(this.handleError)
  );
```

RxJS Operator: `shareReplay`



Shares its input Observable with other subscribers

Replays the defined number of emissions on subscription

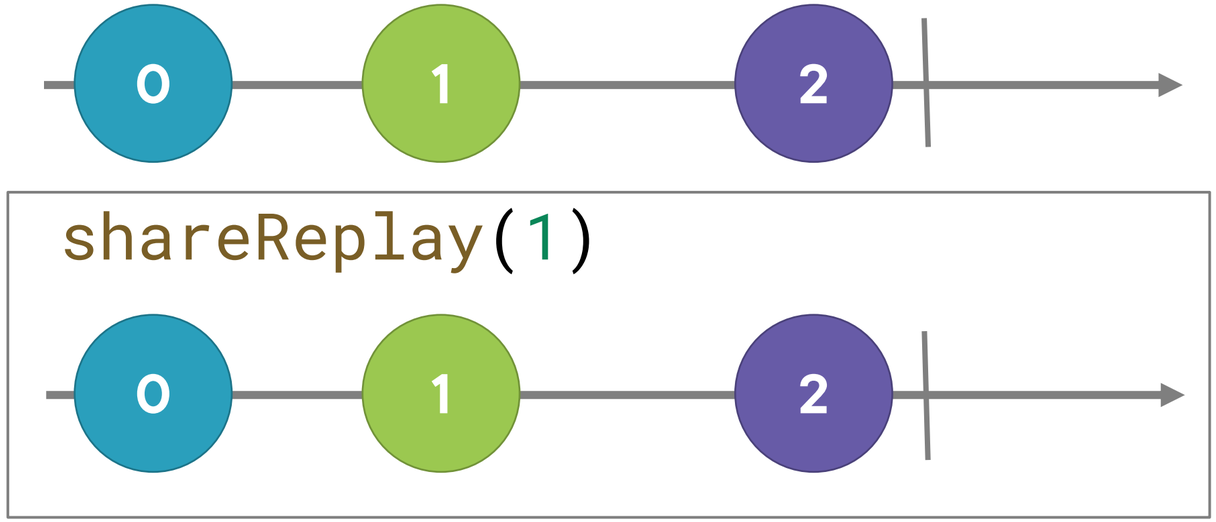
```
shareReplay(1)
```

Used for

- Sharing Observables
- Caching data in the application
- Replaying emissions to late subscribers

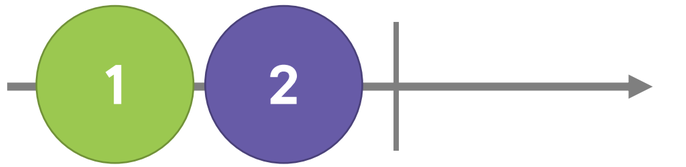
Marble Diagram: shareReplay

```
a$ = interval(1000)
  .pipe(
    take(3),
    shareReplay(1)
  )
```

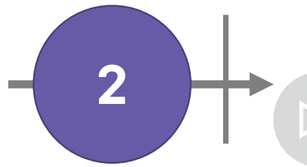



```
a$.subscribe(x =>
    console.log(x));
```

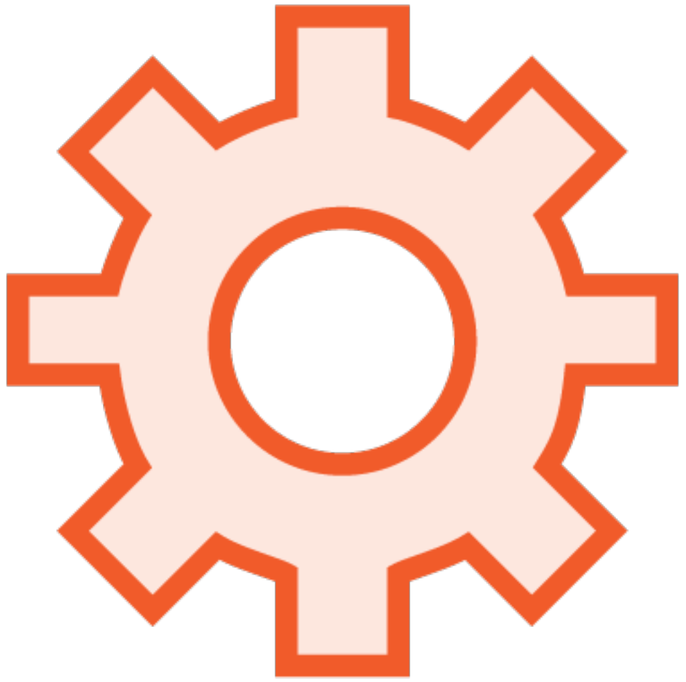

```
a$.subscribe(x =>
    console.log(x));
```




```
a$.subscribe(x =>
    console.log(x));
```



RxJS Operator: shareReplay



`shareReplay` **is a multicast operator**

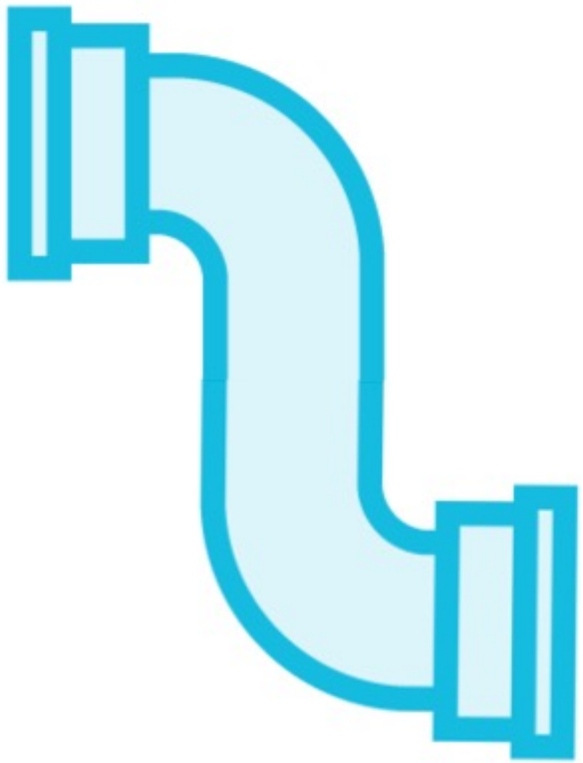
Returns a Subject that shares a single subscription to the underlying source

Takes in an optional buffer size, which is the number of items cached and replayed

On a subscribe, it replays the specified number of emissions

The items stays cached forever, even after there are no more subscribers

RxJS Operator: `share`



Similar to shareReplay

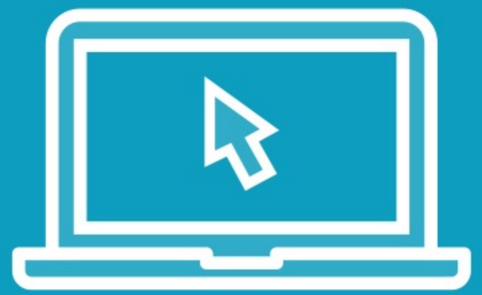
- By default, doesn't have a buffer
- Doesn't replay that buffer

```
share(config)
```

Set of configuration options

```
share({
    connector: () => new ReplaySubject(1),
    resetOnComplete: false,
    resetOnError: false,
    resetOnRefCountZero: false
})
```

Demo



Caching data with `shareReplay`

RxJS Checklist: Purpose of Caching

Retain retrieved data in a service

- No need for another HTTP request
- Reduces redundant computations
- Improves performance and responsiveness

```
selectedProduct$ = combineLatest([
  this.productsWithCategory$,
  this.productSelectedAction$
]).pipe(
  map(([products, selectedProductId]) =>
    products.find(product => product.id === selectedProductId)
  ),
  shareReplay(1)
);
```

RxJS Checklist: Caching

Add shareReplay to any Observable pipeline you wish to share and replay to all new Subscribers

```
productCategories$ = this.http.get<ProductCategory[]>(this.url)
   .pipe(
     tap(data => console.log(JSON.stringify(data))),
     shareReplay(1),
     catchError(this.handleError)
   );
```

RxJS Checklist: Cache Invalidation

Evaluate

- Fluidity of data
- Users' behavior

Consider

- Invalidating the cache on a time interval
- Allowing the user to control when data is refreshed
- Always getting fresh data on update operations

```
private refresh = new BehaviorSubject<void>(undefined);

products$ = this.refresh
  .pipe(
    mergeMap(() => this.http.get<Product[]>(this.url)
      .pipe(
        catchError(this.handleError)
      )
  ));
```

Coming up next...

Higher-order Mapping Operators