



大数据产品部



CONTENTS 目录

1

为什么要使用Typescript

2

Angular框架都有什么

3

异步任务并不是黑魔法

4


神通广大的 Zone.js

5

Angular如何做状态变更

6

RXJS能给项目带来哪些改变



Angular ? angular.js

学习曲线、复杂、困难

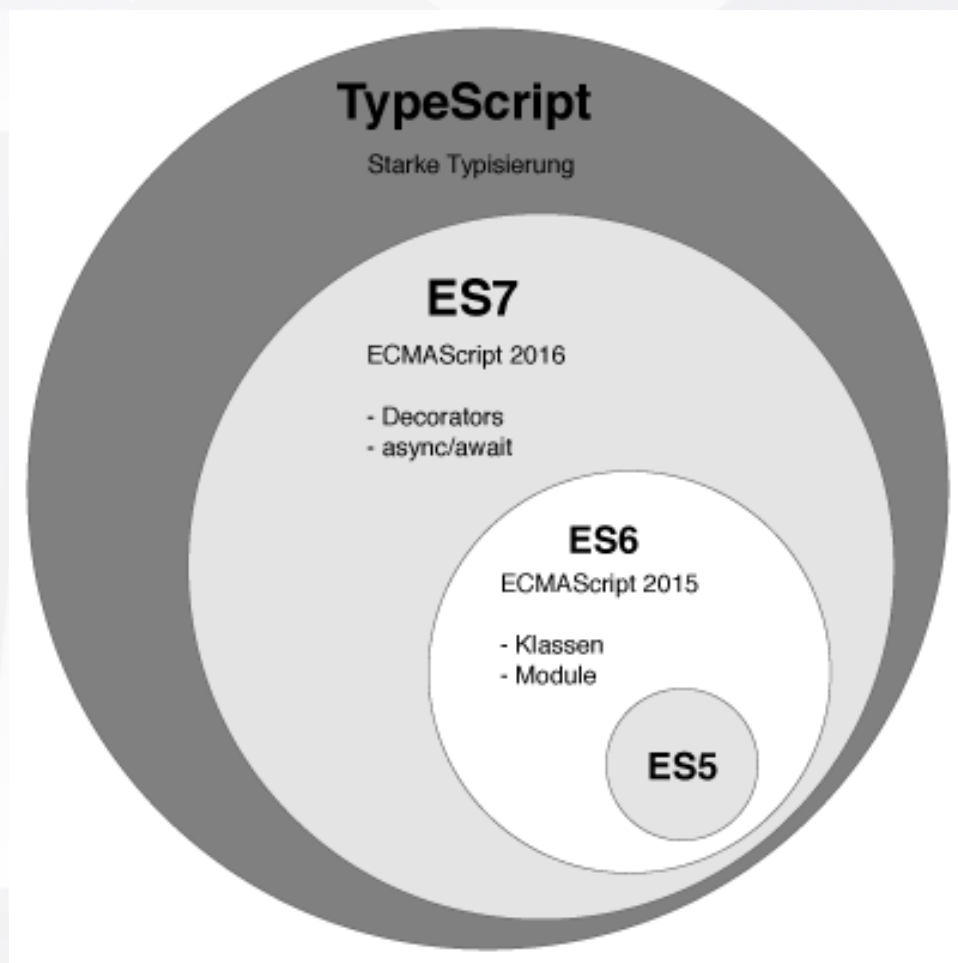
版本变化快

TS增加复杂度

状态管理

8102年了为什么还学Angular

为什么要使用TypeScript



思考：在 js 中如何数据如何描述？

编译时的类型推断 (Type Assertion)

// 支持类型: 布尔、数字、字符、数组、元组、枚举、Any、Void、Null、Undefined、never
boolean, number, string, number[]/Array<number>, [number, string],
enum Color{Red, Green}, any, **null**, **undefined**,

```
export class Hero {  
    name?: string;  
    state?: string;  
}
```

```
const hero: Hero = {  
    name: 'Sam',  
    state: 'active'  
}; // true
```

```
const hero: Hero = {  
    name: 1,  
    state: 'active'  
}; // Error, name property must be string type
```

泛型 & 接口

```
class Queue<T> {  
    private data = [];  
    push = (item: T) =>  
        this.data.push(item);  
    pop = (): T => this.data.shift();  
}  
const queue = new Queue<number>();  
queue.push(0); // true  
queue.push('1'); // error
```

```
interface Point {  
    x: number;  
    y: number;  
}
```

```
class MyPoint implements Point {  
    x: number;  
    y: number;  
}
```

编译时的类型推断 (Type Assertion)

```
errorType.ts x
1
2 export class Hero {
3   name?: string;
4   state?: string;
5 }
6
7 // (property) Hero.name?: string
8 name: 1,
9 state: 'active'
10 }; // Error, name property must be string type
11
```

```
(method) Array<string>.indexOf(searchElement: string, fromIndex?: number): number
Returns the index of the first occurrence of a value in an array.
@param searchElement — The value to locate in the array.
@param fromIndex — The array index at which to begin the search. If fromIndex is omitted, the search starts at index 0.
od.week.i
(index, 1
od.month.indexOf(day);
e(index, 1):
```

```
ngon
}
}
chang
doc
thi
[]
indexOf
join
keys
lastIndexOf
length
map
= 'not angular'
```


|| 声明 (d.ts)

例如es5.d.ts :

```
interface Array<T> {  
    length: number;  
    toString(): string;  
    push(...items: T[]): number;  
    join(separator?: string): string;  
    indexOf(searchElement: T, fromIndex?: number): number;  
    map<U>(callbackfn: (value: T, index: number, array: T[])=> U, thisArg?: any):  
    U[];  
    ...  
}
```


ts支持：

很多常用的 js 库都包含有 d.ts 文件，redux、lodash 等

ts编写：

甚至有一些库就是用typescript编写的，例如rxjs、Angular、deno、vscode等。

工程实践：

vue3.0 正在使用 ts 重写
react社区的 flex/ts 实践等

Angular 框架与渐进式

模块 (ngModule)

组件 (component)

服务 (service/DI)

路由 (router)

指令 (directive)

表单 (form)

管道 (pipe)

动画 (Animation)

构建 (CLI)

测试 (Unit & E2E)

国际化 (i18n)

rxjs

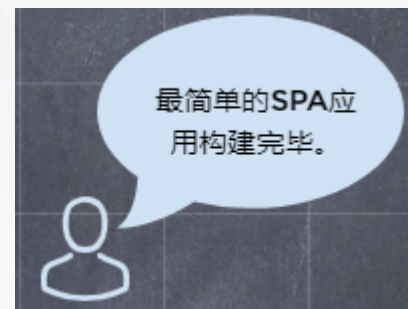
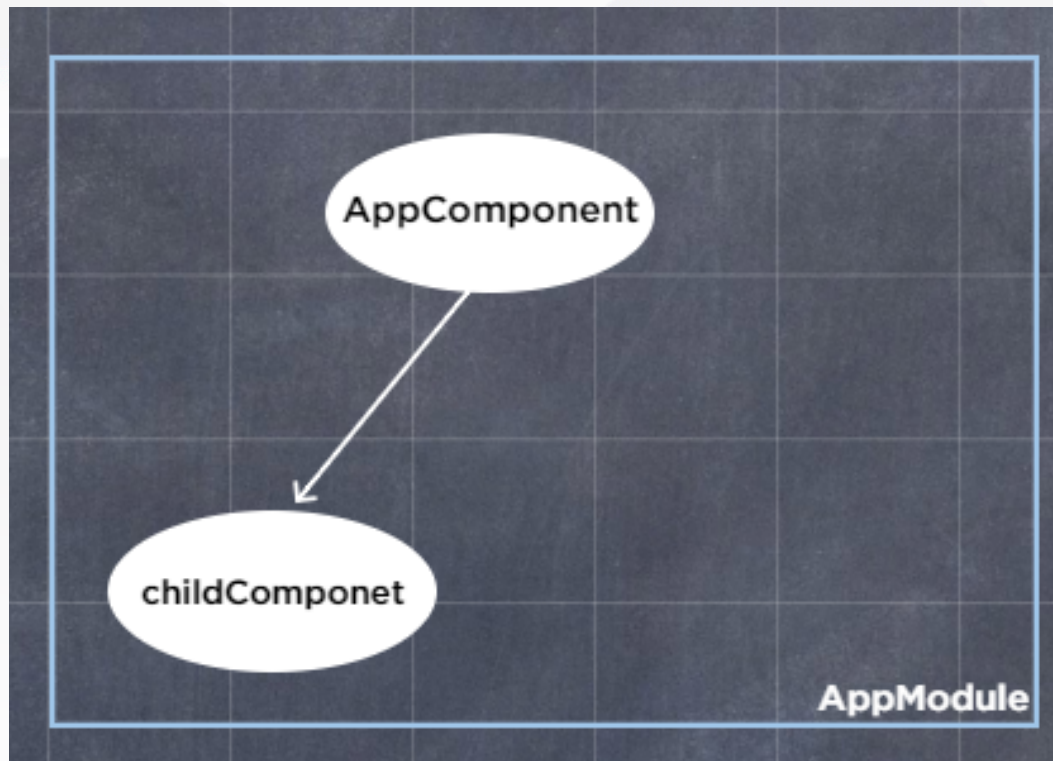
All Platforms

渐进式

NgModule可以理解为一个内聚模块或者一个内聚的功能单元，这个单元里具备自己的生态资源，例如具备组件、指令、服务、管道等。

根Module为AppModule，多个Module组成一个Module树。

Component



Service

SOC原则：通过Service编写逻辑，让组件关注View相关工作。

IOC：Angular帮助管理Service

DI：服务与使用者之间的关系在运行时绑定

```
// Define
@Injectable()
export class xxxService() {}

// Use
@Component()
constructor( private service: xxxService) { }
```

优势：

减少耦合，提高组件的复用性，提升组件的可维护性，易于测试等。

想象一下如下场景：

场景1：组件中需要使用service A，serviceA调用了 service B 和 service C。

场景2：跨域子父组件之间的状态交互。

demo2: <https://stackblitz.com/edit/angular-share-service-demo>

数据何时变化？

button click、form submit

setTimeout、setInterval

XHR / Fetch / jsonp

promise

requestAnimationFrame

框架如何应对变化？

`$watch $digest $apply`

`setState`

`Object.defineProperty()`

Zone.js

Angular 原理 - 变更检测 - Zone

A Zone is an execution context that persists across async tasks. You can think of it as thread-local storage for JavaScript VMs.

执行上下文

异步Hook

堆栈追踪

Angular 原理 - 变更检测 - NgZone

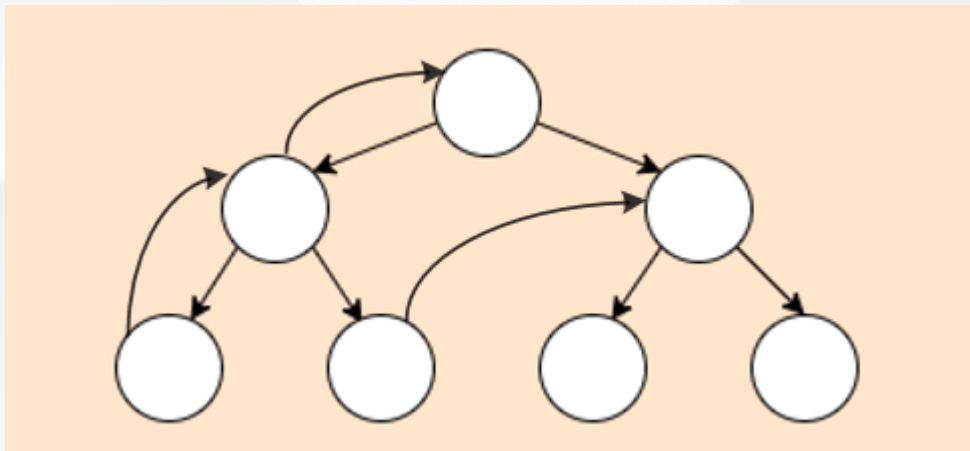
构建在 zone.js 之上

Angular 可以脱离 zone.js 运行

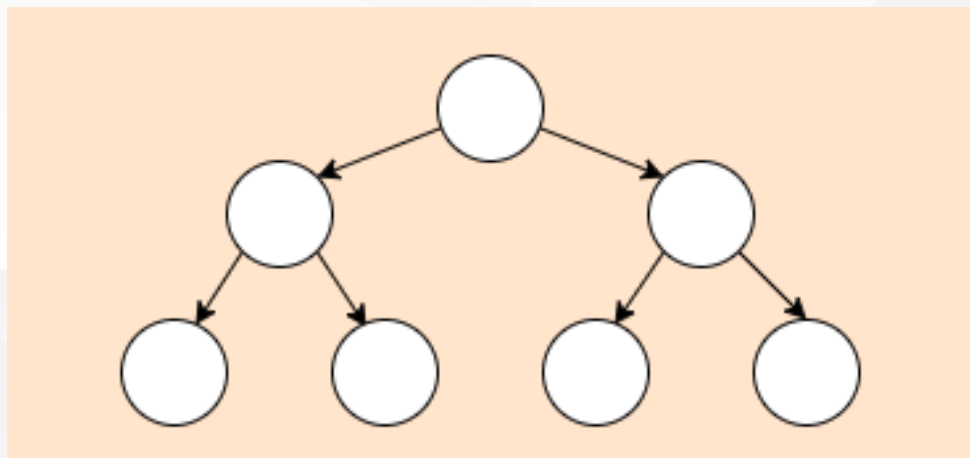
性能优化的手段（后面提及）

Change Detection

Angular 原理 - 变更检测 - NgZone

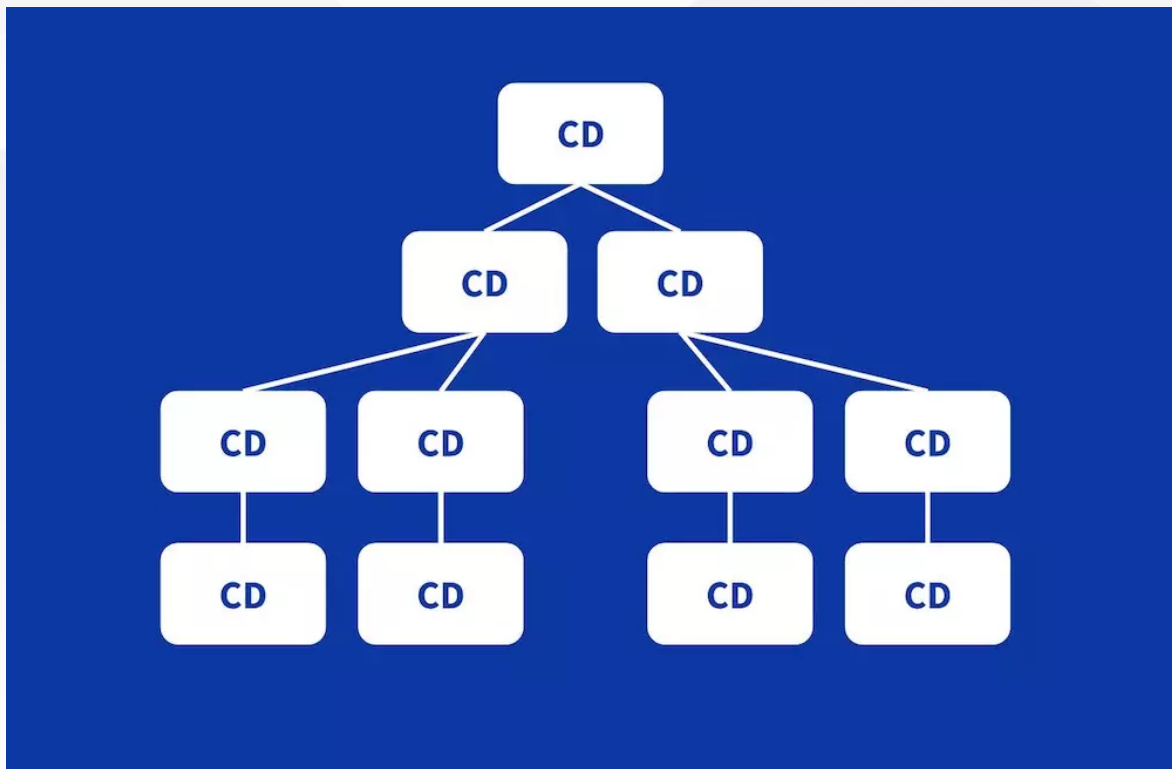


有向环图



单向数据流（树）

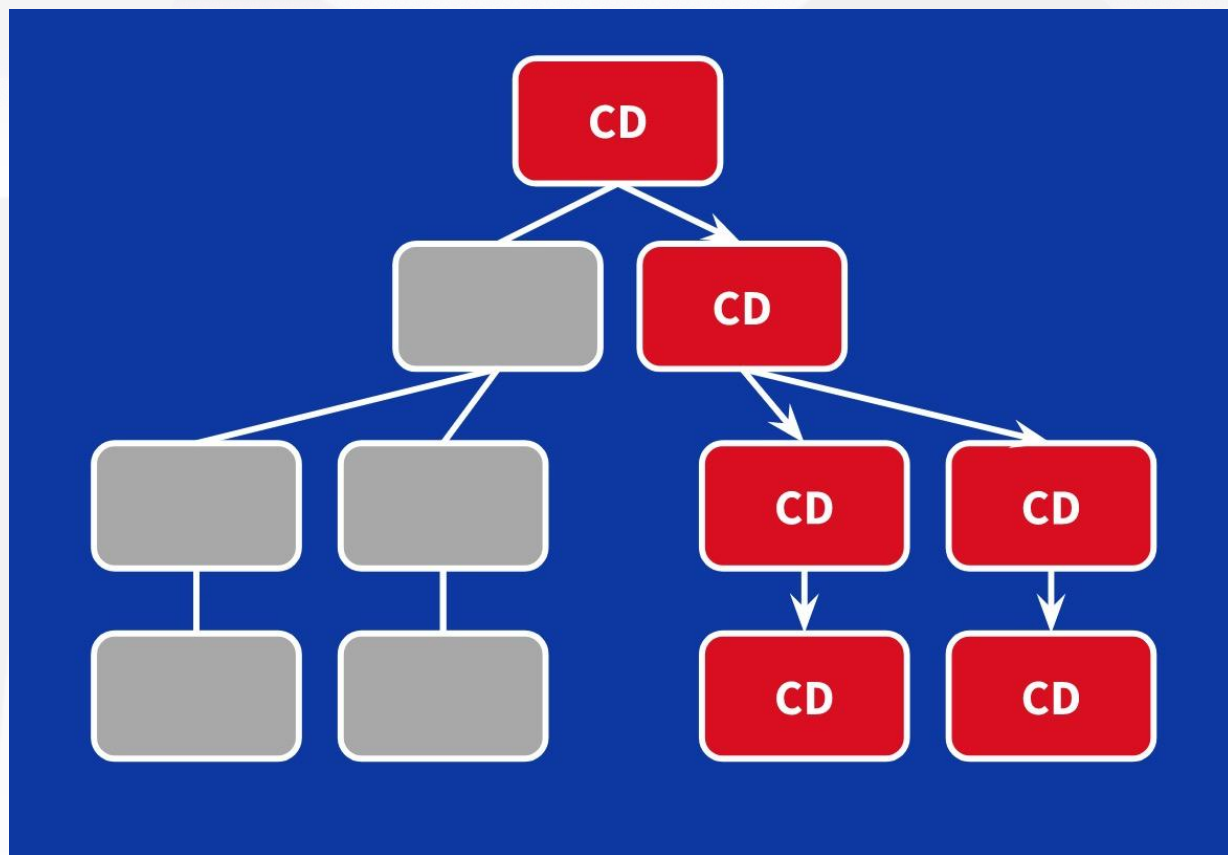
Angular 原理 - 变更检测 - NgZone



```
export const enum ViewState {  
    FirstCheck = 1 << 0,  
    ChecksEnabled = 1 << 1,  
    Errored = 1 << 2,  
    Destroyed = 1 << 3  
    ...  
}
```

auto trigger with Zone.js / ngZone

Angular 原理 - 变更检测 - NgZone



```
ChangeDetectorRef{  
  checkNoChanges  
  detach  
  detectChanges  
  markForCheck  
  reattach  
}
```

```
@component({  
  selector: 'xxx',  
  templateUrl: 'xxx',  
  changeDetection:  
    ChangeDetectionStrategy.OnPush  
})
```

demo3 : <https://stackblitz.com/edit/angular-share-first-know-cd>

demo4 : <https://stackblitz.com/edit/angular-share-cd-tree-demo>

Angular 原理 - 变更检测 - 优化 ?

immutable object

observable

ngZone

demo5 : <https://stackblitz.com/edit/angular-share-ngzone>

函数式 & 响应式 编程

声明式

纯函数

数据不可变性



数据流

变化传播

高阶函数

```
// (i + i) * (i+1 + i+1)
for(var i=0; i<3; i++) {
  console.log(calculate(plus, i));
}
```

```
function calculate(func) {
  return func(i) * func(i+1);
}
```

```
function plus(x) {
  return x + x;
}
```

```
map<U>(callbackfn: (value: T, index: number, array: T[]) => U, thisArg?: any):
U[];
```

```
some(callbackfn: (value: T, index: number, array: T[]) => boolean, thisArg?:
any): boolean;
```

```
reduce(callbackfn: (previousValue: T, currentValue: T, currentIndex: number,
array: T[]) => T): T;
```

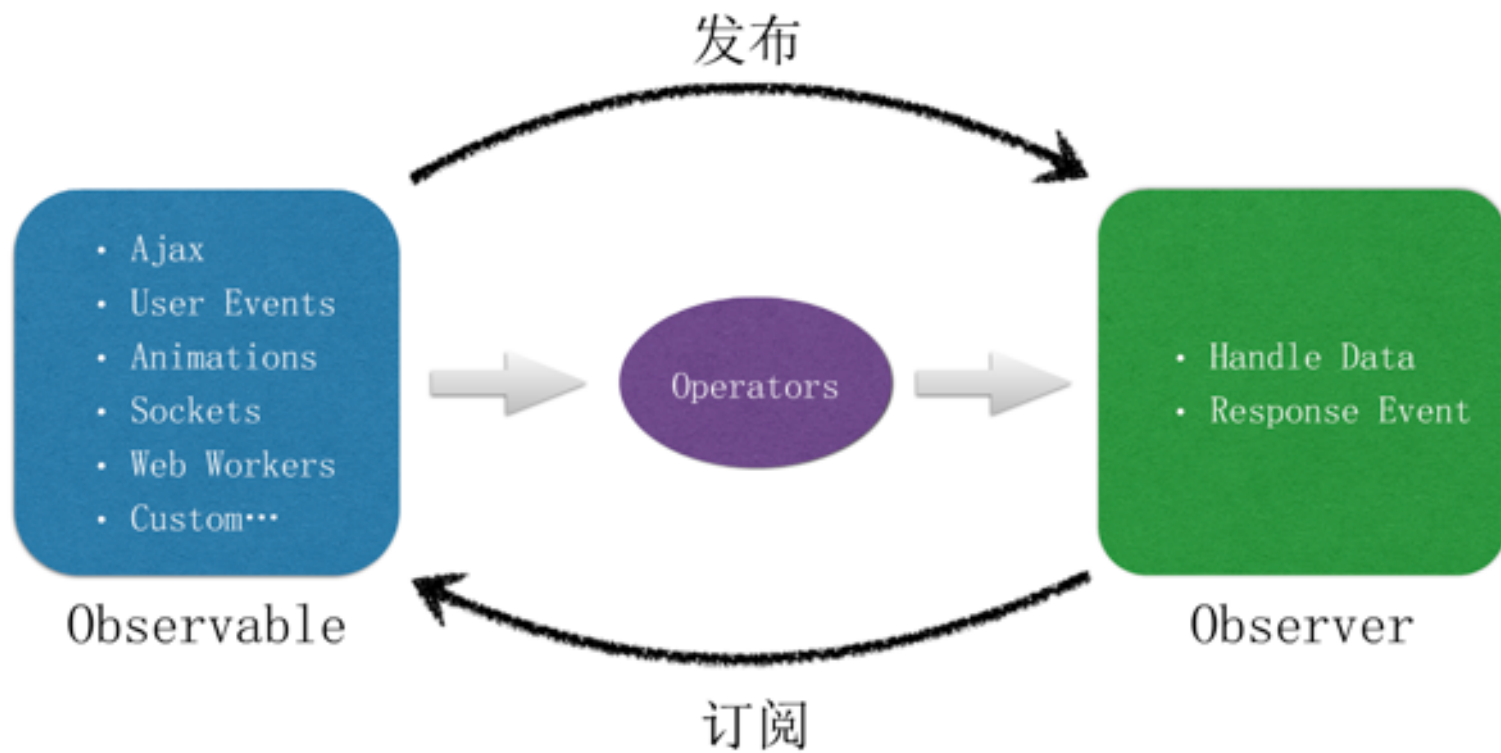
RXJS 中场景的模式



观察者模式

迭代器模式

观察者 / 发布订阅模式



迭代器模式

```
class IteratorFromArray {  
  constructor(arr) {  
    this._array = arr;  
    this._cursor = 0;  
  }  
  next() {  
    return this._cursor < this._array.length ?  
      { value: this._array[this._cursor++], done: false } :  
      { done: true };  
  }  
}
```

这个例子，是否可以优化？

举例：

mousemove 没有必要每次都捕获当前 Box 所在的位置
过滤无效操作

优化手段，非 rxjs 下如何做？

RXJS 操作符

创建 : of、from、ajax、fromPromise、fromEvent、range ...

转化 : map、pluck、scan、mergeMap、windowTime ...

过滤 : filter、take、skip、distinct、debounce、throttle ...

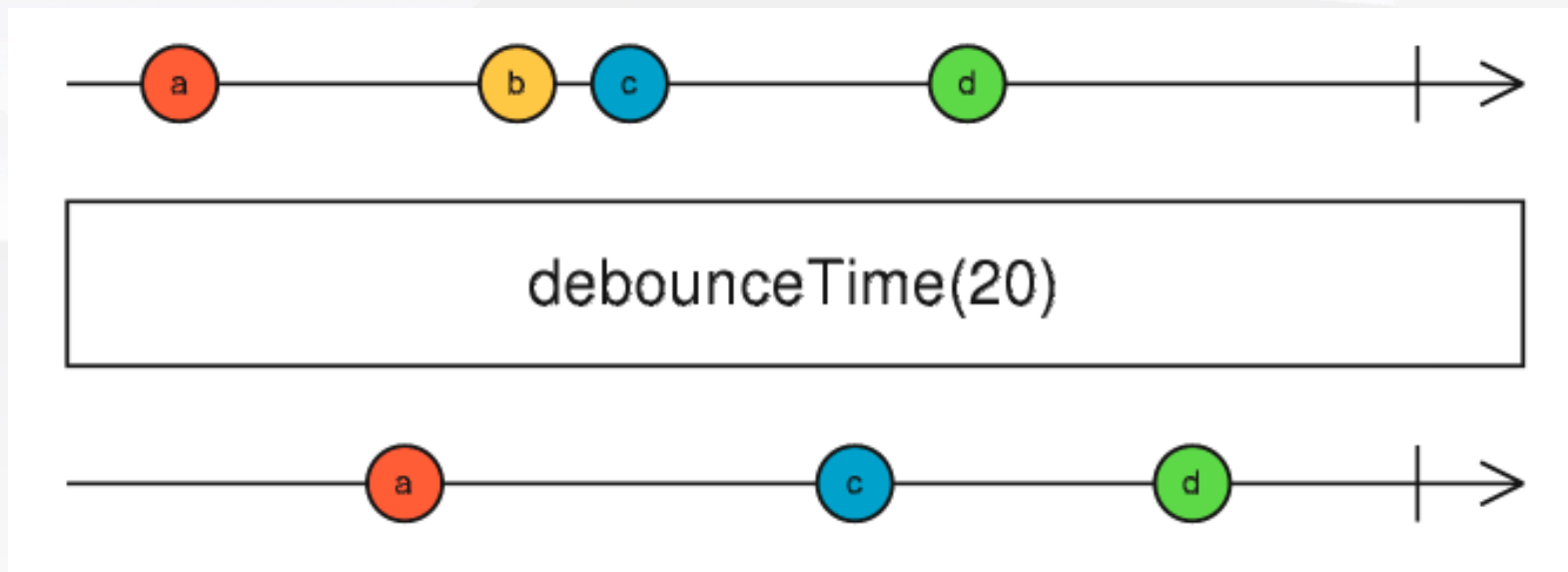
合并 : concat、forkJoin、zip、race、switch ...

多播 : multicast、publishLast ...

辅助 : count、every、reduce、max、min ...

错误处理 : retry、catch、finally ...

RXJS 弹珠图



```
import { Observable, fromEvent } from 'rxjs';
import { debounceTime } from 'rxjs/operators';
const source$ = fromEvent(document, 'click');
const result$ = source$.pipe(
  debounceTime(500)
);

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


Observable & Observer

```
function create(subscriber) {  
  var observable = {  
    subscribe: (observer) => {  
      subscriber(observer);  
    }  
  }  
  return observable;  
}
```

```
var observable = create((observer) => {  
  observer.next(1);  
  observer.next(2);  
  observer.next(3);  
});
```

```
var observer = {  
  next: (value) => console.log(value),  
  complete: () => console.log('complete')  
};
```

```
observable.subscribe(observer);
```

RXJS 特征 & 应用

延迟执行

多播

异步流程处理

(串联、多值、订阅、取消、错误处理)

框架：

EventEmitter

Http

Async

Router

Reactive Forms

状态管理

demo8 : stackblitz.com/edit/angular-share-service-rxjs-demo

结束话题

CLI

定制化

Ivy

Bazel

schematics

elements

可插拔

跨平台



感谢大家~