02

OPEN ORIENTED

凹 凸 实 验 室

RxJS 入门介绍

Youshan

RxJS概念

RxJS 概念

RxJS 是 Reactive Extensions for JavaScript 的缩写，起源于 Reactive Extensions，是一个基于可观测数据流在异步编程应用中的库。RxJS 是 Reactive Extensions 在 JavaScript 上的实现，而其他语言也有相应的实现，如 RxJava、RxAndroid、RxSwift 等。学习 RxJS，我们需要从可观测数据流(Streams)说起，它是 Rx 中一个重要的数据类型。

RxJS 概念

02

把RxJS当做一个针对事件的Lodash

需求：搜索功能

监听文本框的输入事件，将输入内容发送到后台，最终将后台返回的数据进行处理并展示成搜索结果。

RxJS 概念


```
<input id="text"></input>
<script>
    var text = document.querySelector('#text'),
        timer = null,
        currentSearch = '';

    text.addEventListener('keyup', (e) =>{
        clearTimeout(timer)
        timer = setTimeout(() => {
            // 声明一个当前所搜的状态变量
            currentSearch = '书';

            var searchText = e.target.value;
            $.ajax({
                url: `search.qq.com/${searchText}`,
                success: data => {
                    // 判断后台返回的标志与我们存的当前搜索变量是否一致
                    if (data.search === currentSearch) {
                        // 渲染展示
                        render(data);
                    } else {
                        // ..
                    }
                }
            });
        },250)
    })
</script>
```

RxJS 概念

02

使用RxJS基于可观测数据流实现

RxJS 概念


```
var text = document.querySelector('#text');
var inputStream = Rx.Observable.fromEvent(text, 'keyup')
                      .debounceTime(250)
                      .pluck('target', 'value')
                      .switchMap(url => Http.get(url))
                      .subscribe(data => render(data));
```

RxJS 概念


```
var text = document.querySelector('#text');
var inputStream = Rx.Observable.fromEvent(text, 'keyup')
                    .debounceTime(250)
                    .pluck('target', 'value')
                    .switchMap(url => Http.get(url))
                    .subscribe(data => render(data));
```

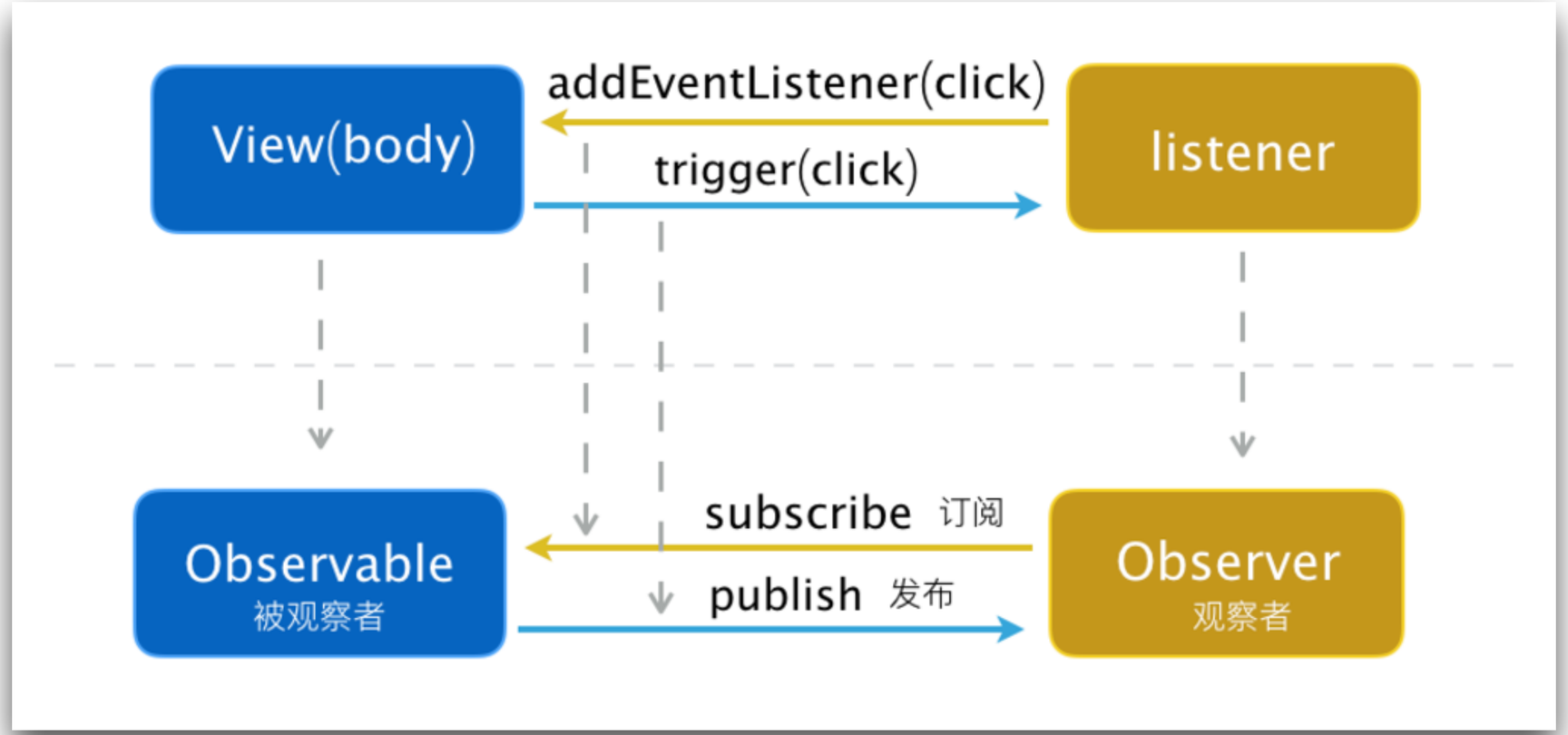
Observable 可观察对象

Observer 观察者（数据的消费者）

Operator 操作符（返回一个新的可观察对象）

RxJS 概念

观察者模式



迭代器模式

```
var iterable = [1, 2];

var iterator = iterable[Symbol.iterator]();

iterator.next(); // => { value: "1", done: false}
iterator.next(); // => { value: "2", done: false}

iterator.next(); // => { value: undefined, done: true}
```

（value 表示返回值，done 表示是否已经到达最后。）

相关资料

RxJS 相关资料

官方文档：http://reactivex.io/rxjs/

珠宝图：http://rxmarbles.com/

RxJS可视化：https://rxviz.com/

案例讲解

THANKS

FOR YOUR WATCHING