响应式编程与 RxJS

@被删

前端入门

前端深入理解

前端架构

谈谈依赖和解耦

大型前端项目要怎么跟踪和分析函数调用链

前端构建大型应用

响应式编程在前端领域的应用

异步数据流

响应式编程在前端领域

比较其他技术

热观察与冷观察

合流

其他使用方式

VSCode 源码解读：事件系统设计

VSCode 源码解读：IPC通信机制

在线文档的网络层设计思考

补齐Web前端性能分析的工具盲点

前端监控体系搭建

其实在几年前因为 Angular 的原因接触过响应式编程，而这些年的一些项目经验，让我在再次回顾响应式编程的时候又有了新的理解。

什么是响应式编程

响应式编程基于观察者模式，是一种面向数据流和变化传播的声明式编程方式。

异步数据流

响应式编程常常用在异步数据流，通过订阅某个数据流，可以对数据进行一系列流式处理，例如过滤、计算、转换、合流等，配合函数式编程可以实现很多优秀的场景。

除了天然异步的前端、客户端等 GUI 开发以外，响应式编程在大数据处理中也同样拥有高并发、分布式、依赖解耦等优势，在这种同步阻塞转异步的并发场景下会有较大的性能提升，淘宝业务架构就是使用响应式的架构。

响应式编程在前端领域

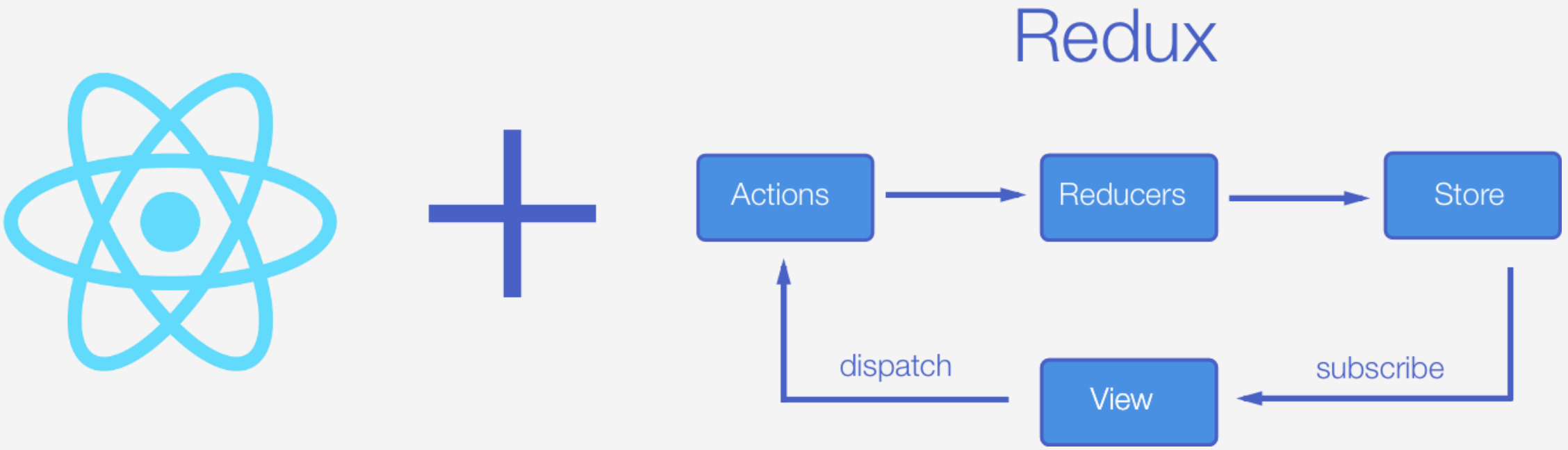
在前端领域，常见的异步编程场景包括事件处理、用户输入、HTTP 响应等。对于这类型的数据流，可以使用响应式编程的方式来进行设计。

不少开发者基于响应式编程设计了一些工具库，包括 Rxjs、Mobx、Cycle.js 等。其中，Rxjs 提供了基于可

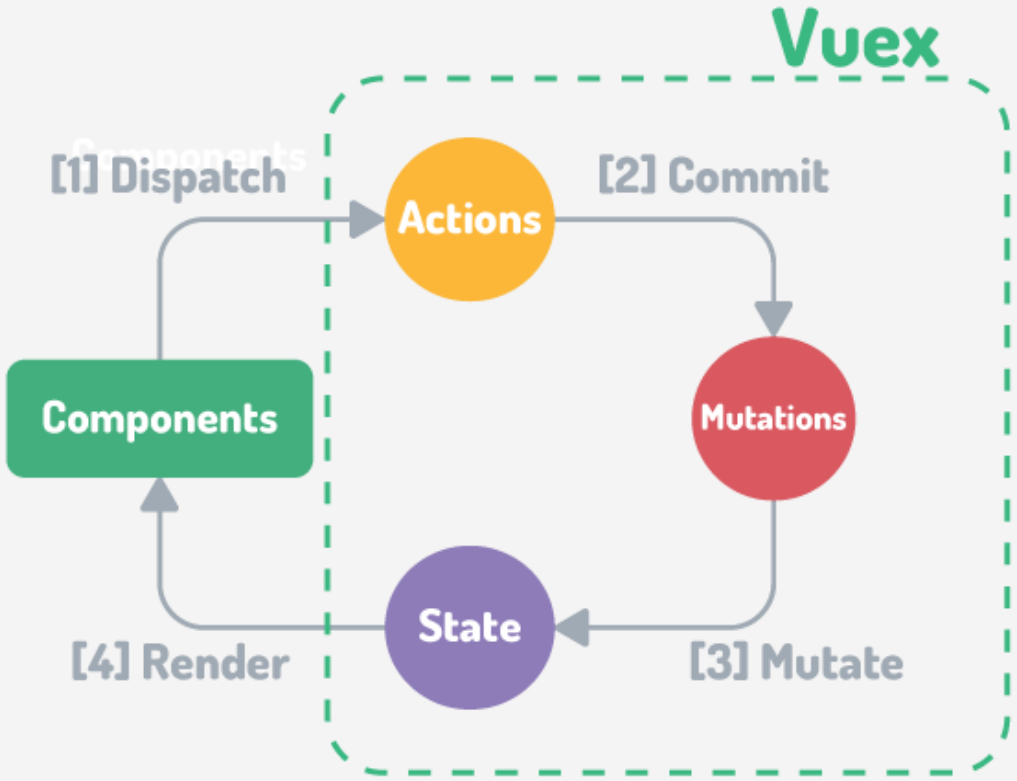
1

什么是响应式编程

前端状态管理



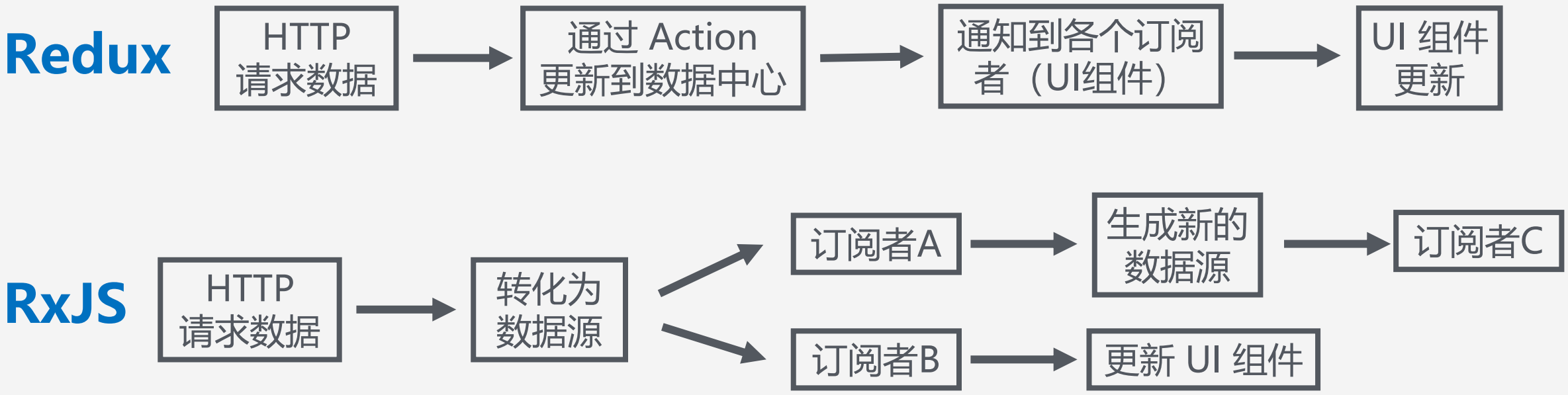
前端状态管理



本质上是通过集中管理数据（数据中心）的方式来维护应用状态

响应式编程区别在哪

Redux vs RxJS



响应式编程

- 响应式编程常常用在异步数据流
- 通过订阅某个数据流，可以对数据进行一系列流式处理（过滤、计算、转换、合流等）
- 配合函数式编程可以实现很多优秀的场景
- 在大数据处理中也同样拥有高并发、分布式、依赖解耦等优势（淘宝业务架构）

2

RxJS 的使用

RxJS 核心概念

- Observable (可观察对象): 表示一个可调用的未来值或事件的集合
- Observer (观察者): 监听由 Observable 提供的值
- Subscription (订阅): 表示 Observable 的执行，用于取消 Observable 的执行
- Operators (操作符): 采用函数式编程风格的纯函数 (pure function)，使用像 map、filter、concat、flatMap 等这样的操作符来处理集合

RxJS 核心概念

- Observable (可观察对象)
- Observer (观察者)
- Subscription (订阅)
- Operators (操作符)

```
var button = document.querySelector('button');
Rx.Observable.fromEvent(button, 'click')
  .throttleTime(1000)
  .map(event => event.clientX)
  .scan((count, clientX) => count + clientX, 0)
  .subscribe(count => console.log(count));
```

防抖、节流、计数器、阈值管理等 利器

RxJS 的使用场景?

HTTP 请求与重试

```js
import { ajax } from "rxjs/ajax";
import { map, retry, catchError } from "rxjs/operators";

const apiData = ajax("/api/data").pipe(
  // 可以在 catchError 之前使用 retry 操作符。它会订阅到原始的来源可观察对象，此处为重新发起 HTTP
  retry(3), // 失败前会重试最多 3 次
  map((res) => {
    if (!res.response) {
      throw new Error("Value expected!");
    }
    return res.response;
  }),
  catchError((err) => of([]))
);

apiData.subscribe({
  next(x) {
    console.log("data: ", x);
  },
  error(err) {
    console.log("errors already caught... will not run");
  },
});
```

用户输入

```js
const observable = Rx.Observable.fromEvent(input, "input") // 监听 input 元素的 input 事件
  .map((e) => e.target.value) // 一旦发生，把事件对象 e 映射成 input 元素的值
  .filter((value) => value.length >= 1) // 接着过滤掉值长度小于 1 的
  .distinctUntilChanged() // 如果该值和过去最新的值相等，则忽略
  .subscribe(
    // subscribe 拿到数据
    (x) => console.log(x),
    (err) => console.error(err)
  );
// 订阅
observable.subscribe((x) => console.log(x));
```

有没有更好玩的?

热观察与冷观察

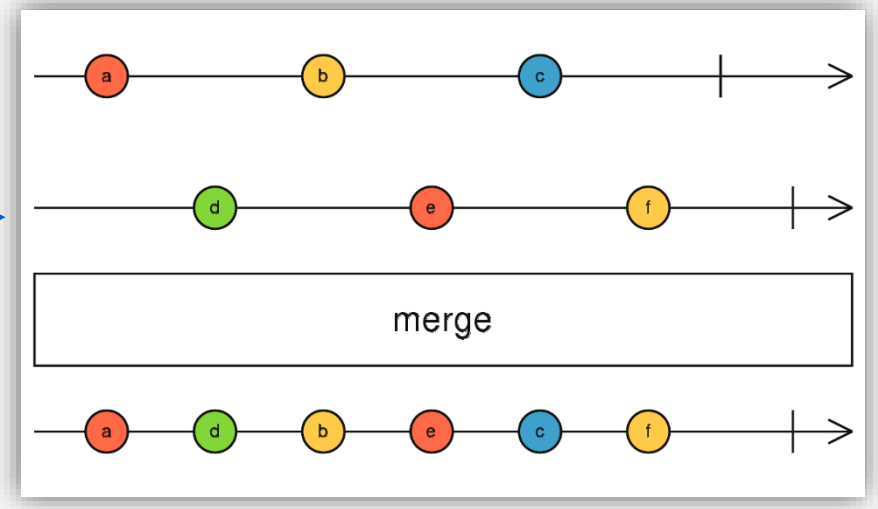
- Cold Observable，可以理解为点播（电影），我们打开的时候会从头播放
- Hot Observable：可以理解为现场直播，我们进场的时候只能看到即时的内容

```
let liveStreaming$ = Rx.Observable.interval(1000).take(5);

liveStreaming$.subscribe(
    data => console.log('subscriber from first second')
    err => console.log(err),
    () => console.log('completed')
)

setTimeout(() => {
    liveStreaming$.subscribe(
        data => console.log('subscriber from 2nd second')
        err => console.log(err),
        () => console.log('completed')
    )
}, 2000)
// 事实上两个订阅者接收到的值都是 0,1,2,3,4，此处为冷观察
```

```
let publisher$ = Rx.Observable.interval(1000).take(5).publish();

publisher$.subscribe(
    data => console.log('subscriber from first minute',data),
    err => console.log(err),
    () => console.log('completed')
)

setTimeout(() => {
    publisher$.subscribe(
        data => console.log('subscriber from 2nd minute', data),
        err => console.log(err),
        () => console.log('completed')
    )
}, 3000)

publisher$.connect();
// 第一个订阅者输出的是0,1,2,3,4，而第二个输出的是3,4，此处为热观察
```

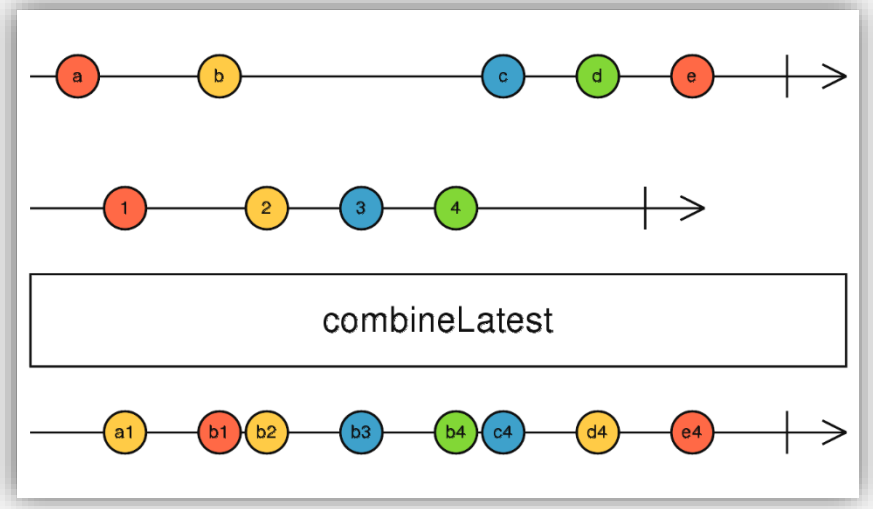
两种合流方式

1. merge 合流

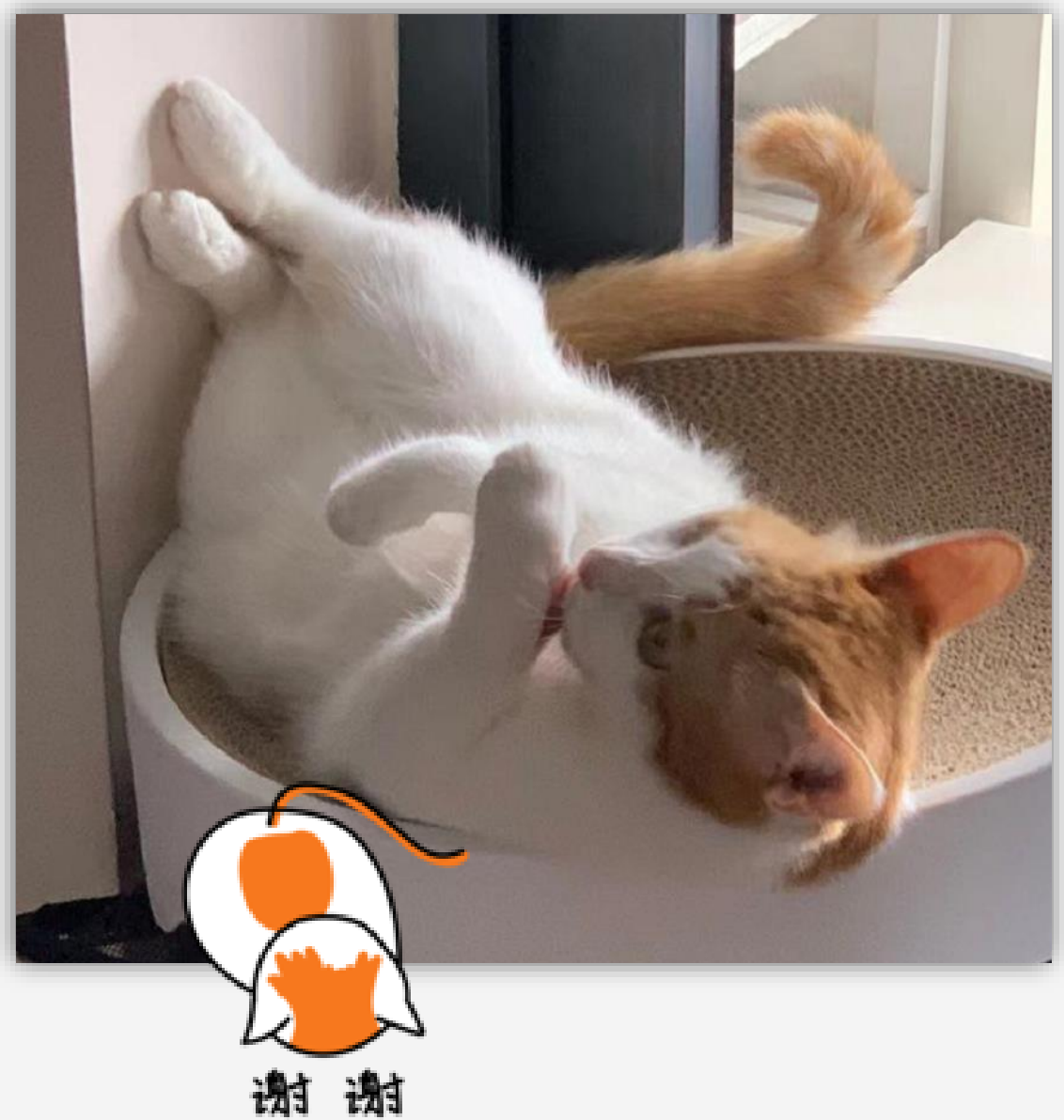


- 群聊天
- 聊天室
- 公众号订阅

2. combine 合流



在 Excel 中，
通过函数计算 A1 和
B2 两个格子的相加



Github: godbasin
@被删