D2 前端技术论坛
D2 FRONTEND TECHNOLOGY FORUM
精心

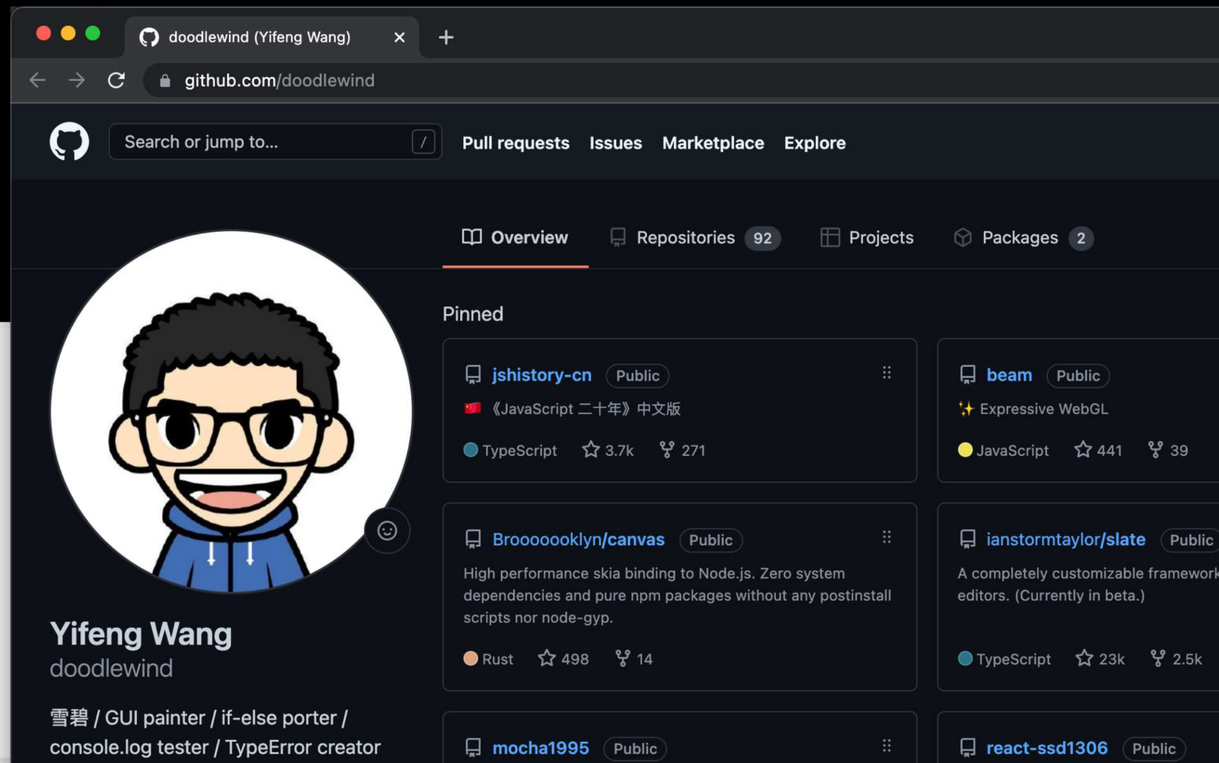
第十六届D2前端技术论坛

CRDT 实时协作技术在稿定编辑器中的应用

王译锋（雪碧）

About Me...

- 2016 年毕业于中国科技大学，从事前端工作至今
- 喜欢分享前端杂学，交友 ID doodlewind



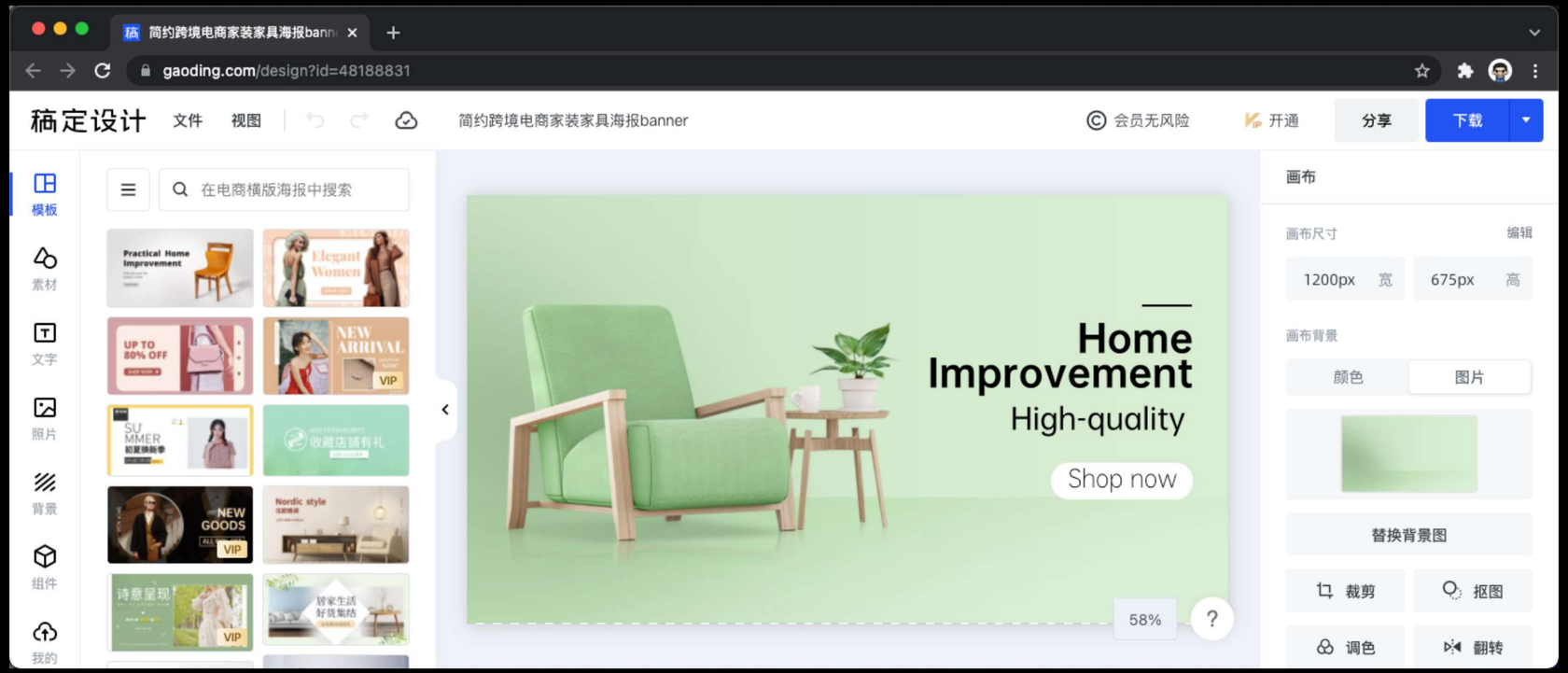
Contents
目录

01

背景知识

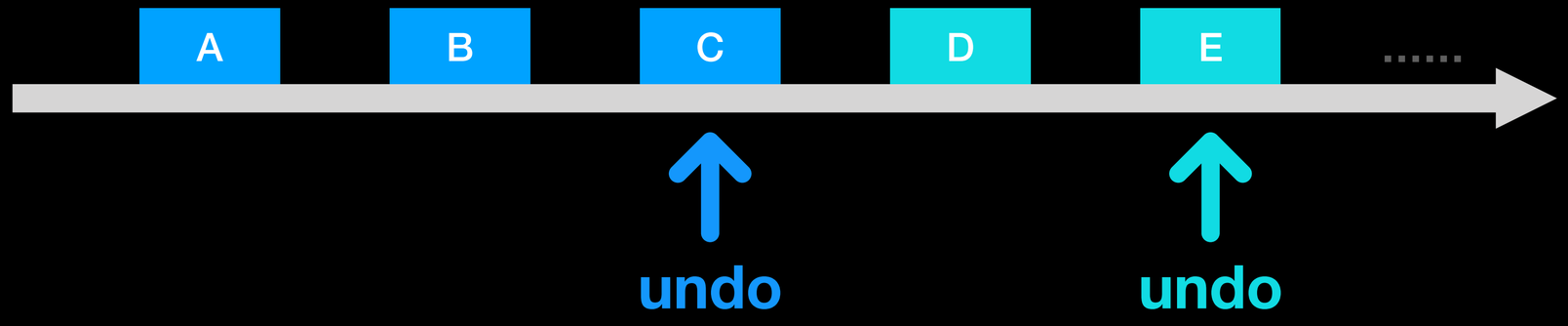
业务背景

- 稿定平面设计编辑器：每月服务数万合作设计师与上百万用户的生产力工具
- 区分 C 端个人版与 B 端企业版，后者支持团队协作
- 支持富文本编辑，但主要数据结构仍为树形，元素均为具备较多属性的节点



从状态管理到实时协作：前置知识概述

- 常规「状态管理」在软件工程研究中的抽象：Event Sourcing 模式
  - 将 **model** 操作变为可序列化的 **operation** 数据，Redux 即为典型
  - 构建 operation 栈，在 undo / redo 时移动栈指针应用 operation 即可
- 实时协作场景下的挑战
  - 由于网络延迟，分布式产生的 operation 在各客户端可能以不同顺序合并
  - 由于用户仅可撤销自身改动，故需能任意「移除」operation 栈底的某项



如何使朴素的 Event Souring 机制适应协作场景？

- 引入类似 git 的 rebase 机制
  - 服务端唯一确定 operation 顺序，在各客户端 rebase 后应用
  - 需依赖中心节点，亦不便支持离线编辑
- 引入对 operation 数据的变换（**Operational Transform，即 OT**）
  - 将 operation 在服务端变换，客户端只需应用变换后的 operation
  - 需引入中心节点，有较多经典案例但工作量大
- 引入特殊的 model 数据结构（**Conflict-free replicated data type，即 CRDT**）
  - 每个用户具备唯一 UID，其每次操作均有 version vector（类似逻辑时间戳）
  - 每个对象字段均关联 version vector，合并时依逻辑时序 last win 即可

02

CRDT 库 Yjs 的接入实践

CRDT 库 Yjs 概述

- 源于 YATA 算法，较 RGA 等业界原有 CRDT 算法获得了显著改进
- 最坏时间复杂度为合并远程 insert 时的 O(N^2)，其余场景均不超过 O(logN)
- 社区已为其实现了 Slate、Quill、ProseMirror 等编辑器的 binding 插件

CRDT	LOCAL		REMOTE	
	INS	DEL	INS	DEL
WooT	$O(H^3)$	$O(H)$	$O(H^3)$	$O(H)$
WooTO	$O(H^2)$	$O(H)$	$O(H^2)$	$O(H)$
Logoot	$O(H)$	$O(1)$	$O(H \cdot log(H))$	$O(H \cdot log(H))$
RGA	$O(H)$	$O(H)$	$O(H)$	$O(log(H))$
YATA	$O(log(H))$	$O(log(H))$	$O(H^2)$	$O(log(H))$

Table 1: Worst case time-complexity analysis, adapted from [1].

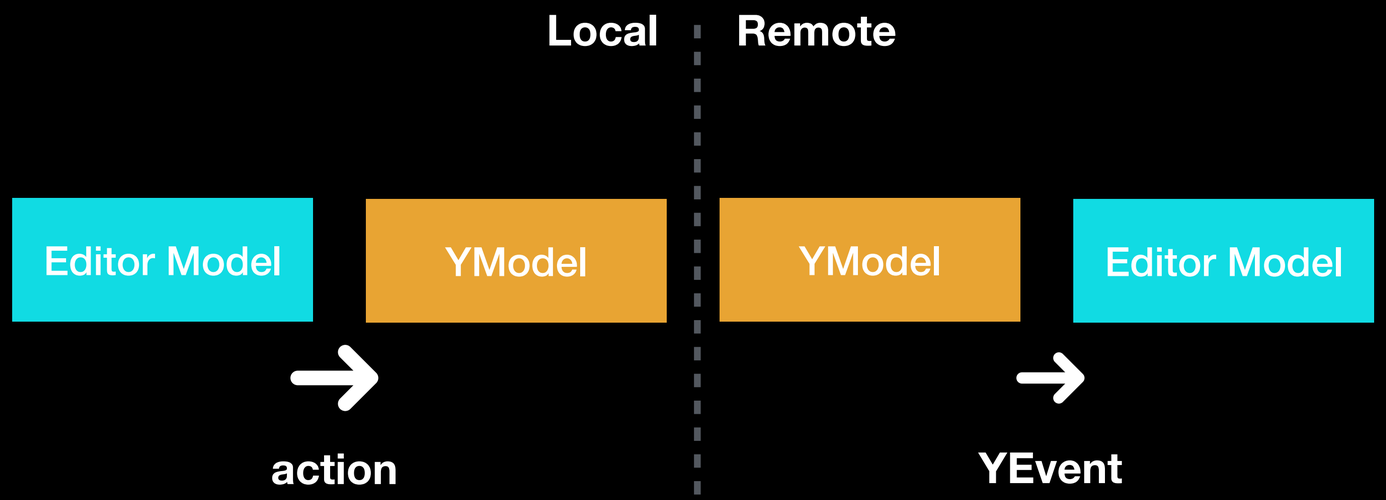
Yjs 接入实践

- 其 API 类似 model 层，提供 YMap、 YArray、 YText 等 Shared Type
- YModel 较难完全透明地与常规 model 同步生命周期，故建议维护 binding 层
- YDoc 可在各客户端实例化，并关联 WebsocketProvider 透明地同步状态

```
import * as Y from 'yjs'

const yDoc = new Y.Doc()
const yElements = yDoc.getArray('elements')

const yElement = new Y.Map()
yElement.set('left', 100)
yElement.set('top', 100)
yElement.set('type', 'image') // 可使用非 YText 的字符串
yElement.get('left') // 注意此时还拿不到结果

yElements.push([yElement]) // 关联到 YDoc 节点下
yElement.get('left') // 状态已可获取，并在 YDoc 实例之间共享
```

状态双向同步方案

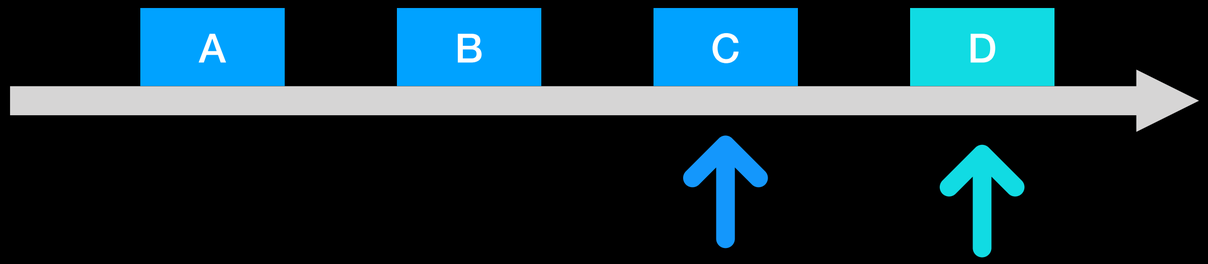
- 为编辑器建模 action
  - 各类 action 无需可序列化，相当于简易 operation
  - 在 binding 层根据 action 同步修改 YModel 即可
- 本地 ➡️ 远程：基于编辑器 action 修改 YModel
- 远程 ➡️ 本地：基于 YEvent 修改编辑器 model




```
let isRemoteUpdating = false

async function remoteObserver(events) {
  isRemoteUpdating = true
  // 根据远程更改的 YEvent 修改编辑器 model
  // ...
  isRemoteUpdating = false
}

function commit(action) {
  if (isRemoteUpdating) return

  // 忽略本地更改触发的 YEvent，避免死循环
  yStore.unobserveDeep(remoteObserver)

  // 在此根据编辑器 action 修改 YModel
  // ...

  yStore.observeDeep(remoteObserver)
}
```

历史记录

- 核心设计准则
  - 用户只能撤销自己的改动
  - 用户从状态 A 独立撤销 N 次之后再重做 N 次，要能回到 A
- Y.UndoManager 符合上述准则
  - 为每个用户维护一个本地的 undo 栈
  - 对 undo 前后两份 model 状态做 diff，将该 diff 以 YEvent 形式发送



独立撤销 C 理想中应获得 A B D
但实践中应获得 A B C D C’
且只需保证 C C’ 等价于空操作
即可保证 undo 后 redo 的 A B C D C’ C 仍为 A B C D

后端配套方案

- CRDT 无需中心节点，大幅降低了后端开发成本
- 配套后端方案的关键点
  - 基于 Yjs 配套的 y-websocket 搭建房间，自动广播房间内各客户端改动
  - 可将 YEvent 的 Uint8Array 更新数据缓存至 Redis，令服务端实例横向扩展
  - 在 Node 端对 YModel 做序列化操作，即可保存文档版本
  - 可参考社区库 Hocuspocos 封装供鉴权、房间新建等业务使用的生命周期钩子

03

状态同步方案优化

面向嵌套节点树的状态同步优化

- 朴素实现
  - 一对一映射 YModel 结构（YModel + YArray 表达力与 JSON 等价）
  - 更新时以元素为粒度整体更新节点
- 改进
  - 支持树形结构扁平化与 fractional-indexing
  - 支持字段级的 diff 增量更新

树形结构扁平化

- 问题
  - 朴素实现会将元素成组操作建模为「先删除子元素，再整体添加新组」
  - 对「A **成组** -> B **解组** -> A **撤销** -> B **撤销**」路径，会出现重复元素
- 优化方案
  - 将 YModel 建模为扁平的 Map<UUID, YModel> 结构
  - 为元素维护 parentId 和 index 字段，在 reparent 操作时更改 parentId 即可
  - 优化后可支持元素的 reparent 与 shift 语义

Fractional-indexing

- 问题
  - 扁平化后需维护元素 index 字段，更新范围大且可能在撤销后出现重复
- 优化方案
  - A B C 元素下标不再为 [0, 1, 2] 形式，而是 { A: 0.25, B: 0.5, C: 0.75 } 结构
  - 在 B C 之间插入 D 时，获得 { A: 0.25, B: 0.5, C: 0.75, D: 0.625 }

```
import {generateKeyBetween} from 'fractional-indexing';

const first = generateKeyBetween(null, null); // 'a0'
// Insert after 1st
const second = generateKeyBetween(first, null); // 'a1'
// Insert after 2nd
const third = generateKeyBetween(second, null); // 'a2'
// Insert before 1st
const zeroth = generateKeyBetween(null, first); // 'Zz'
// Insert in between 2nd and 3rd. Midpoint
const secondAndHalf = generateKeyBetween(second, third); // 'a1V'
```

字段级 diff 增量更新

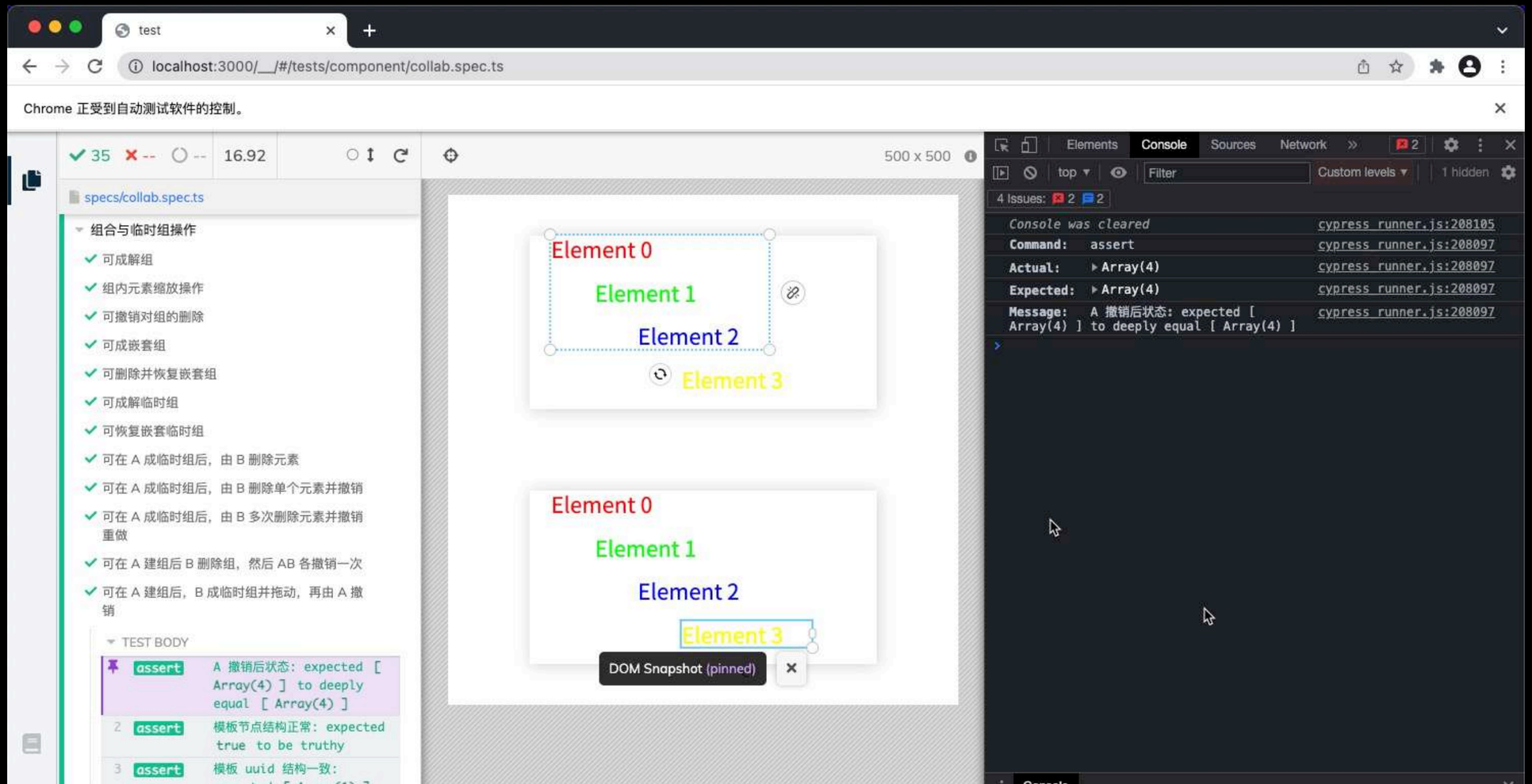
- 问题
  - 若每次同步均全量更新 YModel 字段，每次 undo 时的粒度也只能精确到元素
  - 对「A **更改颜色** -> B **更改文字** -> A **撤销**」路径，将额外撤销 B 的改动
- 优化方案
  - 精确到元素字段级的自动 diff
  - 仍提交 change_element 类型的 action，附带上更新完成后的元素实例
  - 对比 action 中的元素新状态与相应 YModel 的旧状态，仅更新 diff 变化的字段

04

自动化测试与 benchmark

自动化测试与 benchmark

- 开发环境：基于 Cypress + Vite 的 TDD 环境
- 性能测试：基于 Node 的 CLI 压测工具



自动化测试：面向协作场景的测试工具

- 基于 Cypress Component Testing 搭建本地调试页
  - 支持完全可视化的 DOM 环境，无需 Node 侧模拟 JSDOM
  - 支持 Vite，具备无等待的开发体验
  - 支持 mount 单个 Vue 组件实例，较传统的 visit URL 形式 E2E 显著更轻
- 使用两个本地 Editor 实例，以交换二进制更新数据的形式通信，无需 WebSocket
- 可对比两个实例的 model 状态与 parentId 等字段合法性，及时发现异常状态

性能测试：使用 Node 模拟多用户场景

- 在 Node 端用 y-websocket 实例连入房间，模拟常见编辑器 YModel 更新
- 内存占用较高，2M 模板 100 客户端每 500ms 局部更新 5 分钟，需 3GB 内存
- 房间人数增长后，新用户连入时有显著延迟（对当前 50 人量级足够使用）

05

总结

总结

- CRDT 方案已在前端实时协作领域具备强竞争力
- CRDT 的设计需离散数学理论，但工程落地所需的仍是基础数据结构知识
- 技术方案应具备理论可靠性，否则正确实现的方案仍可能有重大问题

Thanks