协同编辑方案调研总结

协同编辑算法介绍

协同编辑并非新兴技术，它已经有着数十年的发展历史。谈到协同编辑，常常会涉及两类算法：OT(Operational Transformation) 和 CRDT(Conflict-free Replicated Data Type)。

OT算法：OT算法通过操作记录和转换来实现数据一致性，适用于实时协同编辑。OT算法的优点是能实时反映用户操作并处理并发冲突，但需要中心化服务器进行协同调度，可能不适用于大规模分布式系统。

CRDT算法：CRDT算法通过数据结构的合并来实现数据一致性，不需要服务器解决冲突。CRDT算法的优势在于去中心化、性能、灵活性和可用性。随着 CRDT 的逐步发展，以及YJS框架将其性能优化到了难以置信的地步，新的产品也更多地开始基于 CRDT 开发协同编辑功能，例如 Figma、Miro 等。

Yjs框架：Yjs是一个基于CRDT的开源协同编辑框架，专为web上构建协同应用程序设计。Yjs作为一个数据结构，通过CRDT进行数据更新和增量同步，支持多种编辑器接入，并且处理冲突、光标等核心功能。

基于YJS的实现方案

数据转换&解决冲突

Yjs 中的冲突解决数据结构被称为 Shared Types。在富文本编辑器中我们使用JS对象表示应用的状态，然后通过Binding层，将其转换成YJS的 Shared Types。然后进行协同冲突处理，就能实现多人应用的协同编辑能力的支持。

```
// JS 对象
const context = {
    id: 'id',
    users: ['user1', 'user2']
}
```

转换成Shared Types：

```
import * as Y from 'yjs';
```

```
2
3  const doc = new Y.Doc();
4  const yData = doc.getMap('data');
5
6  yData.set('id', 'id');
7
8  const yArray = new Y.Array();
9  yArray.push(['user1', 'user2']);
10 yArray.set('users', yArray);
```

数据更新

在富文本编辑器中，可以通过 `EditorState` 获取应用的状态，但我们不能直接将全量数据设置到 YJS 的 Binding 层，一方面性能会很差，另一方面可能会导致将没有更新的配置识别为已更新。因此，我们需要将已同步的实体模型和现在更新后的实体模型进行差异比较，只对真正更新的配置进行处理。

获取差异数据后，再对 Binding 模型进行增量更新。基本操作如下：

1. 监控到 `EditorState` 状态的变化，获取全量数据。
2. 对比前后两次数据的差异，得到更新数据。
3. 对 Binding 层模型进行增量更新。
4. 通过 Provider 将更新分发出去。
5. 远端在接收更新后，Binding 层反向更新 `EditorState` 数据。

Provider

CRDT 方案本身和网络通信是解耦的，我们可以选择任意方案进行通信，只要能保证更新数据成功的同步到远端即可。YJS 也提供了多种更新通信的能力，在Yjs的语义里面，被称为 Provider。比如：

- 基于 websocket 的 y-websocket
- 基于 webrtc 的 y-webrtc

在C-S模式下，可以使用 y-websocket。后端走 node 服务，需要提供WebSocket 服务进行消息分发，同时还需要维护一份协同过程中的最新副本。新用户在加入后能够立即同步最新的副本，以保持与各端状态一致。

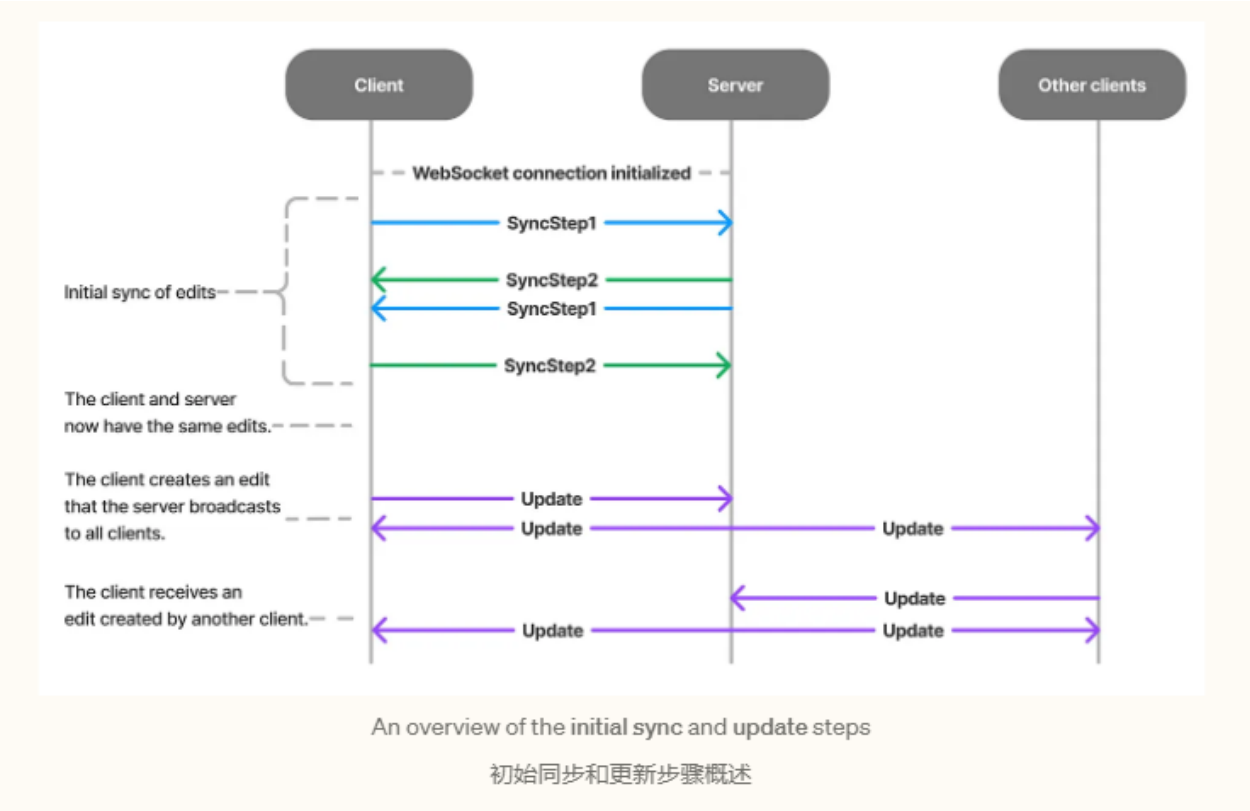
C-S 模式的同步过程：

客户端通过 WebSocket 连接到服务器。连接后，客户端和服务器相互发送对方没有的编辑内容（“初始同步”）。每当客户端生成新的编辑时，它都会将其发送到服务器，然后服务器将其广

播到连接到同一文档的所有客户端。

C-S 同步协议:

在同步协议中，有三种消息类型，客户端和服务器都可以发送：同步步骤1（SyncStep1）、同步步骤2（SyncStep2）和更新（Update）。前两种消息仅用于初始同步，当需要共享任何进一步更新时，则会交换更新消息。



An overview of the initial sync and update steps

初始同步和更新步骤概述

其中:

SyncStep1（同步步骤1）： “请求到目前为止，你拥有但我没有的编辑。”这个过程会有一个消息数据，消息数据是一个状态向量，包含了发送方拥有的插入范围。

SyncStep2 （同步步骤2）： “发送我拥有但你没有的编辑。”发送方过滤其插入集，找出比每个clientId的时钟更新的任何插入，并用对方没有的插入的完整数据进行响应。

Update: 初始同步后，所有进一步的编辑都通过更新消息发送。这些包含与同步步骤2消息相同类型的内容，但仅在初始同步后用于将客户端生成的新编辑共享到服务器，然后从服务器共享到所有客户端。

为什么同步步骤1 → 同步步骤2会重复?

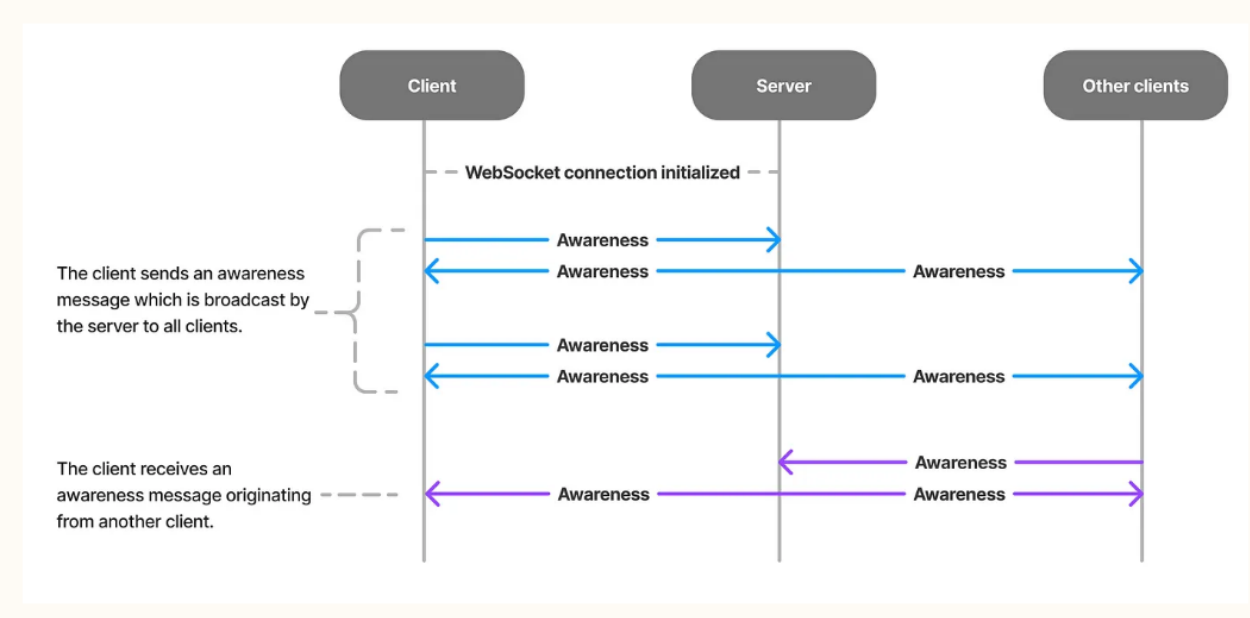
第一轮同步步骤1 → 同步步骤2 **共享服务器到客户端的编辑**，客户端没有这些编辑。在这一轮结束时，客户端拥有服务器拥有的所有编辑，加上只有客户端拥有的编辑。**在第二轮中，客户端向**

服务器发送服务器没有的编辑。此时，客户端和服务器完全同步——他们各自拥有相同的编辑集。

Awareness

在协同编辑过程中，除了保证数据的一致性外，为了优化交互体验，还需要提供诸如：**当前有哪些用户在编辑**，**每个用户在编辑哪部分，光标在哪里** 等等信息。这些信息在 Yjs 的语义中被称之为 Awareness。通常，这些数据量都比较少，因此 Yjs 内部采用了 State-based CRDT 这种处理起来更简单的方式来发送更新。

在默认的 Provider 中，每个客户端每 30 秒向服务器发送一次 Awareness。即使数据本身没有变化（此时只是简单的将内部的 clock+1），如果在过去 30 秒内未从客户端收到感知消息，其他客户端会将其标记为离线。



y-websocket: https://github.com/yjs/y-websocket