学号：20181251046　　密级：

本科毕业论文(设计)

UNDERGRADUATE THESIS

题　　目：现代高交互文档系统的设计与实现

学院名称：计算机科学与技术

专业名称：计算机科学与技术

入校年份：2018 级

学生姓名：胡晨曦

指导教师：王春　讲师

完成时间：2022 年 4 月

淮　北　师　范　大　学

本科毕业论文(设计)

现代高交互文档系统的设计与实现

学院名称：计算机科学与技术

专业名称：计算机科学与技术

学生姓名：胡晨曦

学生学号：20181251046

指导教师：王春 讲师

2022 年 4 月

A Thesis Submitted for the Degree of Bachelor

Design and Implementation of a Modern Highly Interactive Document System

By

Hu Chenxi

Huaibei Normal University

Huaibei, Anhui, P.R. China

April, 2022

淮北师范大学本科生毕业论文（设计）诚信承诺书

本人郑重承诺所呈交的毕业论文（设计）《现代高交互文档系统的设计与实现》，是在指导教师 王春 的指导下严格按照学校和学院有关规定完成的。本人在毕业论文（设计）中引用他人的观点和参考资料均加以注释和说明。本人承诺在毕业论文（设计）选题和研究过程中没有抄袭他人的研究成果和伪造相关数据等行为，若有抄袭行为，由本人承担一切责任。

承诺人：胡晨曦　年级：2018　专业：计算机科学与技术

签　名：胡晨曦

2022年 4月 17日

摘　要

突如其来的疫情，彻底地打乱了人们的生产和生活节奏；在这段人人草木皆兵的时期，想回到办公室与同事面对面沟通，竟变成了一件遥不可及的事情。也正因为此，很多团队及公司都投入了远程协作的怀抱；体验过各种在线协作工具的方便之处后，或许这种新奇的办公方式，也将渐渐成为一种常态。在整套线上协作办公的流程中，文档工具只是其中一环，但也是最不可或缺的一环。与传统的办公文档套件相比，在线文档工具有着文件自动云端保存同步、无需客户端即可使用等特点；同时由于天生在线的属性，在使用多人实时编辑、评论批注等协作功能时，它们也会更加方便。涌现的许多在线文档的工具，进一步方便了人们的在线多人编辑协作。

本论文先探索了目前市场上的主流的多人文本协同编辑方案，针对多人实时编辑所面临的冲突关键技术问题进行了探索，如怎样在多人共同编辑的时候解决冲突问题，怎样更好的设计一个在线文档编辑系统。基于无冲突复制数据类型算法（CRDT）算法思想，利用树形数据结构解决多人协作文本的冲突问题，并对比了与操作转换算法（OT）的差异；TypeScript 是该理论实践所用的主要语音，通过合理分配多个模块和类完成软件设计，并结合课程中所教授的观察者设计模式，降低模块间耦合，提高代码质量，降低了系统的耦合；该系统可以同时适配 C/S 架构和 P2P 架构，并结合具体场景分析了两者的优劣，针对无冲突复制数据类型算法在分布式领域的其他用途进行了展望和思考。

关键词：多人文本协同编辑；无冲突复制数据类型算法；操作转换算法；观察者模式；TypeScript

ABSTRACT

The sudden outbreak of the epidemic has completely disrupted people's production and life rhythm; in this period when everyone is in a state of fear, it has become an unattainable thing to go back to the office and communicate with colleagues face to face. After experiencing the convenience of various online collaboration tools, perhaps this novel way of working will gradually become a norm. In the whole set of online collaborative office process, document tools are only one of the links, but also the most indispensable one. Compared with traditional office document suites, online document tools have features such as automatic cloud preservation and synchronization, and can be used without a client; at the same time, due to their natural online properties, they are also more convenient when using collaborative features such as real-time editing and commenting by multiple people. During this period, many online document tools have emerged to further facilitate online multi-person editing and collaboration.

This thesis first explores the mainstream multi-person text collaborative editing solutions currently available in the market, and addresses the key technical issues of conflicts faced by multi-person real-time editing, such as how to resolve conflicts when multiple people are editing together and how to better design an online document editing system. Based on the idea of conflict-free replicated data type algorithm (CRDT) algorithm, we use tree data structure to solve the conflict problem of multi-person collaborative text, and compare the difference with operation transformation algorithm (OT); TypeScript is the main voice used in this theoretical practice, and the software design is completed through reasonable allocation of multiple modules and classes, and combined with the observer design pattern taught in the course to reduce module The system can be adapted to both C/S and P2P architectures, and the advantages and disadvantages of both are analyzed in the context of specific scenarios, and other uses of the conflict-free replication data type algorithm in the distributed domain are prospected and considered.

Key words： Online Multiplayer Editing; Conflict-free Copy Data Type Algorithm; Operation Conversion algorithm; Observer Model; TypeScript

目　录

1 绪论

1.1 研究的背景

疫情的突然爆发，彻底打乱了人们的生产和生活节奏；在这个人人自危的时期，回到办公室与同事面对面交流，已经成为一件可望而不可即的事情[1]。

在整套在线协作办公流程中，文档工具只是其中的一个环节，但也是最不可或缺的一个。与传统的办公文档套件相比，在线文档工具具有自动云端保存和同步文档的特点，无需客户端即可使用；同时，由于生在网上的属性，它们在使用时将更方便地实现多人实时编辑、评论等协作功能。

新一代信息技术的迅猛发展，深刻影响着我们的工作生活方式。近年，远程办公彻底颠覆了传统的企业管理模式，在线文档作为远程办公软件的重要组成部分也同样迎来了高速发展。

如今，即便市场中已经有了腾讯文档、石墨文档、飞书、语雀和灵犀文档等在线办公产品，但在线文档本身仍面临功能、技术、数据安全、服务、生态等多方面的考验，如数据处理效率、多人协作、二次扩展、系统集成、框架兼容性问题等。

1.2 在线多人协作文档的意义

1.2.1 极大提高工作效率

有工作的地方就会有文件，有文件就会需要不断修改和层层传输。传统的文件办公需要经过很多个环节。举个例子：某部门员工 A 提交策划给领导 B，从提出到确认需要经过以下流程。

A：做完初稿，发送给 B，同时配字“初稿已完成，请您指点”。

B：接收文件，查看，批注，返回给员工 A，同时配字“请按照批注修改”。

A：接受文件，修改，再次发送给领导 B，同时配字“已修改好，请您查看”。

B：并不十分满意，再次批注，返回给员工 A，同时配字“继续修改”。

……

如此反复，最初的 word 文档修改了 N+1 版，整套流程下来，光是提交时说的客套话，就已经换了很多个句式。文件发了无数轮，进展只有一点点，任谁看了不说一声“心好累”。

在线文档提高工作效率的核心是真正做到了协同办公，解决了信息传递零碎且分享不易的问题[2]。

1.2.2 使用简易即用即走

在线文档即用即走、自动保存的特点解决了很多后顾之忧，用户可以多场景、多设备实时同步办公，云端同步储存，再也不需要担心电脑出现意外后前功尽弃，一定程度上减少了工作焦虑。

1.2.3 疫情和灾害大背景

7 月，河南出现百年一遇的特大暴雨灾情。由民间发起的在线文档“待救援人员信息”一时间传遍朋友圈，不断更新着一系列救灾信息，还有众多志愿者自发地核实信息、管理表格，牵动着无数网友的心。

图 1-1 待救援人员信息

这份文档也被称为“救命文档”，搭建了被困群众和外界的联络桥梁。

2019 年疫情前，在线文档还不算流行，人们普遍习惯使用传统的办公文档。年底，一场突如其来的疫情让全员被迫在家办公、学习以及协作，能在线上解决的事情绝不见面。因此，涌现了大量在线办公软件和平台。

1.3 在线多人协作文档的研究现状

1.3.1 在线多人协作文档的市场现状

近年来，文档协作市场不乏玩家入局。2014 年，有道云协作开始上线文档协作功能。2015 年，石墨文档正式上线。而腾讯文档、金山文档、印象笔记等也逐渐成为文档协作的佼佼者。此外，文档协作领域还有小众化的云竹协作、松果文档、一起写，国外更是有 Google Doc、Microsoft 365、Zoho 等。目前国内主流的文档协作产品有腾讯文档、金山文档、石墨文档、有道云笔记、云翔笔记等[3]。

1.3.2 在线多人协作文档的技术研究现状

从技术角度来看，在线、数据处理和多人协作是开发在线文档系统最关键的技术指标。不过，在线和数据处理均已有较成熟的技术方案，实现难度不大。因此，多人协作才是影响在线文档系统易用性的核心要素[4]**Error! Reference source not found.**。

从技术的角度来看，主要有两方面的难点。

一是前端编辑器的复杂交互和良好的编辑体验。

类型	描述	典型产品
L0	1、基于 contenteditable 2、使用 document.execCommand 3、几千～几万行代码	早期的轻量级编辑器
L1	1、基于 contenteditable 2、不用 document.execCommand，自主实现 3、几万行～几十万行代码	CKEditor、TinyMCE Draft.js、Slate 石墨文档、腾讯文档
L2	1、不用 contenteditable，自主实现 2、不用 document.execCommand，自主实现 3、几十万行～几百万行代码	Google Docs Office Word Online iCloud Pages WPS 文字在线版

图 1-2 前端编辑器

L0 的优势在于技术门槛低，短时间内快速研发，但可定制的空间非常的有限；L1 的优势在于站在浏览器的肩膀上，能基本满足大多数的业务场景，但满足不了较为复杂的交互和更好的编辑体验，无法突破浏览器本身的排版效果。L3 的优势是所有的东西都可以定制化，但其难度相当于自研浏览器，数据库[5]。

二是多人实时编辑下的冲突解决，从算法角度来看就是实现某种协作算法

协作算法的一般理论是在用户之间传输变化，以达到“最终一致性/服务”的目的。如果多个用户同时进行更改，他们的文档副本可能会在一段时间内看起来不同，但最终

每个用户的视图都会收敛为相同的视图。这是通过我所说的“基线”完成的; 输入的更改被集成到本地数据中，以形成新的基线，本地更改则基于当前基线发送。

目前的研究下，主要有两种类型的算法被使用，一是操作转换算法，二是无冲突数据类型复制算法。

1.3 本章小结

本章首先介绍了在线多人协作文档产生的时代背景，然后分析了在线多人协作文档的意义，即 1. 极大提高工作效率。2. 使用简易即用即走。 3. 疫情和灾害大背景。接下来分别从市场和技术两方面进行了调研和分析。在技术分析出引出了本论文所涉及的关键算法，无冲突数据类型复制算法，关于这一点会在第二节，研究问题分析中进行详细的探讨和分析。

2 研究问题分析

2.1 理解冲突的背景

大的前提是文档的同步是发送局部的更新，我门每次只发送操作，而不是发送整个文章的内容，事实上，也不应该发送整个文章的内容，否则当文章内容很多的时候，我们每次的细微改动都伴随着整个内容的传输，这显然是一种带宽资源上的浪费，而且会带来糟糕的用户体验，如输入丢失问题[6]。

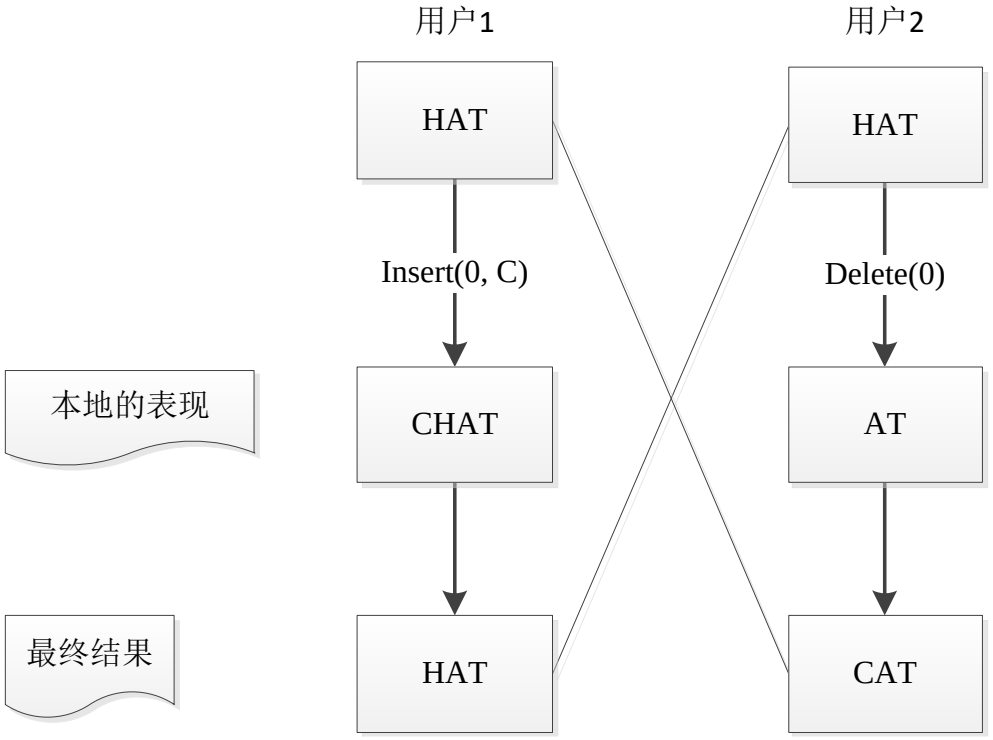

图 2-1 冲突的理解

假设目前有两位用户同时编辑一个文档，他们分别是用户 1、用户 2。

当某个时刻，两人当时的文档内容都是 HAT，User1 产生了想要在 0 的位置添加 C 的炒作，User2 产生了想要删除 0 位置字符的操作。

因此对 User1 来说: HAT → insert(C, 0) → delete(0) → HAT。

因此对 User2 来说: HAT → delete(0) → insert(C, 0) → CAT。

两边编辑同一处内容，产生了不一样的结果，这显然是不符合预期的。

2.2 比对操作转换算法和无冲突数据类型复制算法

操作转换算法和无冲突数据类型复制算法是解决上述冲突问的的主要算法。

2.2.1 操作转换算法

操作转换（OT）是一种支持高级协作软件系统中一系列协作功能的技术。OT 最初是为了在纯文本文档的协作编辑中进行一致性维护和并发控制而发明的。它的能力已经得到了扩展，其应用范围也扩大到了群组撤销、锁定、冲突解决、操作通知和压缩、群组感知、HTML/XML 和树状结构的文档编辑、协作式办公生产力工具、应用程序共享和协作式计算机辅助媒体设计工具。

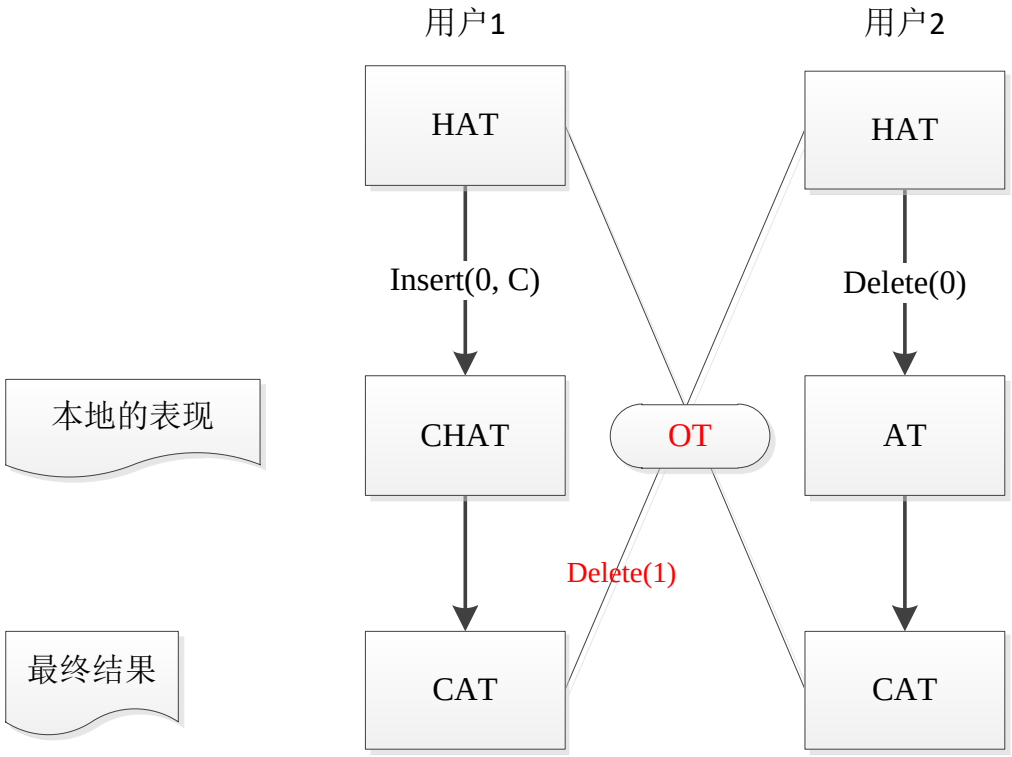

图 2-2 OT 算法的解决方案

利用操作转换的协作系统通常使用复制的文档存储，每个客户端都有自己的文档副本；客户端以无锁、无阻塞的方式对他们的本地副本进行操作，然后将变化传播给其他的客户端；这确保了客户端在其他高延迟的环境中的高响应性，例如互联网。当一个客户端收到从另一个客户端传播的变化时，它通常在执行这些变化之前对其进行转换；转换确保所有站点都能保持与应用相关的一致性标准（不变性）。这种操作模式的结果是，系统特别适合于实现协作功能，如在网络等高延迟环境下的同步文档编辑。

OT 是一种比较并发操作并检测它们是否会导致文档不收敛的算法。如果答案是肯定的，则在应用这些操作之前将对其进行修改(或转换)。

这项技术诞生于 1989 年，其原理是将文本内容统一为以下 3 种类型的操作方式，目的是为用户提供最终一致性实现：

1. retain(n)：保持 n 个字符；

2. insert(str)：插入字符 str；

3. delete(str)：删除字符 str。

在完成上述操作后，OT 算法将正在并发的操作合并转换，以形成新的操作流，并应用在历史版本上，实现无锁化同步编辑。

因此对 User1 来说: HAT → insert(C, 0) → delete(1) → CAT。

因此对 User2 来说: HAT → delete(0) → insert(C, 0) → CAT。

2.2.2 无冲突数据类型复制算法

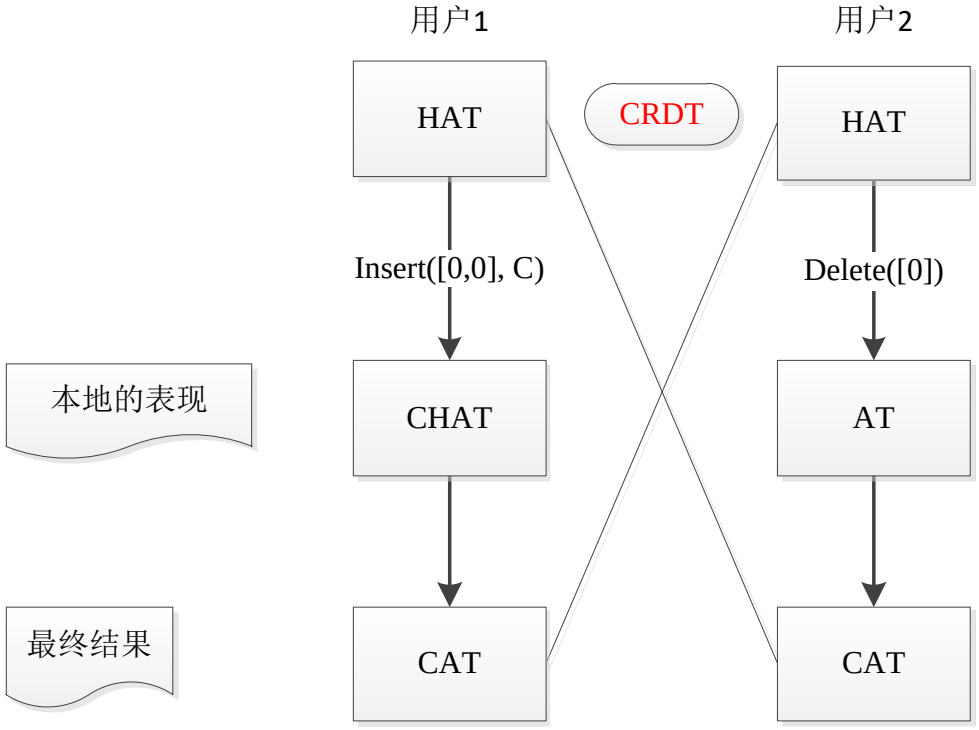

图 2-3 CRDT 算法的解决方案

无冲突复制的数据类型（CRDT）是一种抽象的数据类型，具有明确的接口，被设计为在多个进程中进行复制，并表现出以下特性。

(1)任何副本都可以被修改，而不需要与其他副本协调；

(2)当任何两个副本收到相同的更新集时，它们会以确定性的方式达到相同的状态，通过采用数学上合理的规则来保证状态收敛。

上述是对该算法实现的数据类型的定义，相较于操作转换算法，无冲突数据类型复制算法通过更加复杂的数据结构来完成两件事情。

1. 交换性，这意味着操作来的顺序并不重要，只要有相同的操作集就会有相同的相关的结果

2. 幂等性，重复进行某项操作，删除、增加是无效的，注意操作是具有全局唯一性的。

存在两种不同的 CRDT，一种是基于状态的，其中整个状态在副本之间发送并连续合并。另一种是基于操作的，只有单个操作在副本之间发送。还有具体的实现唯一的算法策略也会不同。

下面接收的方法是基于操作的，利用树形结构构建唯一性的方案。

还通过上面的例子来展示。

因此对 User1 来说: HAT → insert(C, [0,0]) → delete(0) → CAT。

因此对 User2 来说: HAT → delete(0) → insert(C, [0,0]) → CAT。

因为 OT 算法的实现较为复杂和 CRDT 更广法的算法场景，论文设计采用了 CRDT 和树形结构的方法来实现对在线多人编辑文档系统冲突问题的解决。

2.3 本章小结

本章节是整篇论文的核心部分。

首先通过一个具体的场景来解释为什么会出现冲突问题，然后比对了市场上主要存在的两种解决方案操作转换算法和无冲突数据类型复制算法，并结合具体的图演示了算法是如何工作和帮助解决冲突问题的，操作转换算法算法需要有一个中心的服务器节点来进行冲突解决，并把解决后的情况进行下发。而无冲突数据类型复制算法通过利用“相对位置”构建每个字符的唯一性标识来解决冲突问题，这种相对位置不仅对唯一性删除有帮助，在进行添加的时候，同样可以根据这个相对位置寻找到需要添加进入的位置。

此外，还去拓宽视野，了解了其他的可能解决方案，但在在线多人编辑文档这样的高交互性的业务场景，大多数是无法满足的，最终，本设计采用了无冲突数据类型复制算法的解决方案来进行实现。

相对位置的设计从整体来看，如同一个树形的数据结构，这对算法的实现有了更高的要求。

3 系统的开发技术介绍

本章节简要介绍下在开发现代高交互文档系统中所用到的开发依赖和工具，方便其他研究者进行复现和运行。

3.1 版本控制工具

总体来说现代高交互文档系统是一个组合多项 Web 技术和无冲突数据类型复制算法的实践项目，项目开发过程中，考虑到面临的困难在研发初始阶段是不太确定的，因此引入了 git 这样的版本控制工具来辅助，方便当某一方案不满足需求或者开发成本较大的时候，方便回溯到某个版本。。

3.2 类型管理工具

在构建无冲突数据类型复制算法的时候，最重要的一点就是确定好与之相关的数据结构，但原生的 JavaScript 并不具有类型的能力，出于项目严谨性和工程的考虑，引入 TypeScript 来进行类型约束。

TypeScript 起源于 JavaScript 在微软以及客户中开发大型应用中遇到的缺点。处理复杂 JavaScript 代码带来的挑战使他们需要自定义工具来简化组件开发流程。

TypeScript 开发者寻求一种不破坏现有标准兼容性和跨平台支持的解决方案。知道 ECMAScript 标准为未来基于类编程提供支持后, TypeScript 开发便基于此方案。这形成了包含一组新的语法扩展的一个 JavaScript 编译器，一个基于此提案的超集，可将 TypeScript 语法编译为常规的 JavaScript。从这个意义上来讲，TypeScript 是 ECMAScript 2015 预期内容的预览版本。提案中未包括的可选静态类型被添加到了 TypeScript 中，有助于促进工具和 IDE 支持[7]。

3.3 工程化相关工具

如在版本控制工具的介绍中所说，现代高交互文档系统的组成，是多项 Web 技术和无冲突数据类型复制算法的实践项目的结合，因为要引入多项 Web 工具并处理其依赖关系，因此基于工程的考虑引入了 pnpm 这样的现代包管理工具。

与其相关的重要依赖管理配置文件如下。

在 package.json 中

```
  "scripts": {
```

```
    "start": "ts-node src/main.ts",
    "dev": "nodemon --watch 'src/**/*.ts' --exec ts-node src/main.ts",
    "build": "tsc",
    "test": "jest --coverage",
    "open": "open coverage/lcov-report/index.html",
    "example": "parcel example/index.html -p 5000"
},
"devDependencies": {
    "@types/codemirror": "^5.60.5",
    "@types/jest": "^27.4.0",
    "@types/node": "^17.0.18",
    "@types/react": "^17.0.39",
    "@types/react-dom": "^17.0.11",
    "@types/simplemde": "^1.11.8",
    "@typescript-eslint/eslint-plugin": "^5.12.0",
    "@typescript-eslint/parser": "^5.12.0",
    "eslint": "^8.9.0",
    "eslint-config-prettier": "^8.3.0",
    "eslint-plugin-prettier": "^4.0.0",
    "jest": "^27.5.1",
    "nodemon": "^2.0.15",
    "parcel": "^2.3.1",
    "prettier": "^2.5.1",
    "stream-browserify": "^3.0.0",
    "ts-jest": "^27.1.3",
    "ts-node": "^10.5.0",
    "typescript": "^4.5.5"
},
"dependencies": {
    "eventemitter3": "^4.0.7",
    "react": "^17.0.2",
```

```
    "react-dom": "^17.0.2",
    "simplemde": "^1.11.2",
    "treeflex": "^2.0.1"
  }
```

为了项目可以更加健壮稳定，也为了设计无冲突数据类型复制算法的目标产出更加明确，引入 Jest 作为单元测试工具，一方面可以提前约定好描述，想好预期要求再进行具体编码工作，另一方面在项目引入新的需求和设计后，亦可以快速测试已有的功能是否出现异常。

Jest 是一个由 Facebook 公司维护的 JavaScript 测试框架，由 Christoph Nakazawa 设计和构建，专注于简单性和对大型网络应用的支持。它适用于使用 Babel、TypeScript、Node.js、React、Angular、Vue.js 和 Svelte 的项目。对于第一次使用测试框架的用户来说，Jest 不需要大量的配置。

3.4 编辑器工具

现代高交互文档系统把研究的重点放在无冲突数据类型复制算法的实现和编辑器数据模型与算法的结合上，编辑器方面借用了已经十分成熟的 codemirror，因为算法需要对操作的相关信息进行传输，但浏览器原生支持的 input 或者 textarea 在相关的 event 事件中并不能拿到具体是什么操作，因此这里借用了 codemirror 来完成对用户输入的信息的收集[8]。

另外 codemirror 具有良好的拓展性和定制接口，当需求有更多定制化的内容时候，可以更加流畅的引入新的需求和变动。

3.5 本章小结

在具体介绍系统的设计与实现之前，本章提前阐述了在系统开发中将会使用的几项工具，包括版本控制工具、类型管理工具、工程化相关工具，编辑器工具，这些工具为项目的顺利推进都起到了不可或缺的作用。

4 系统的设计和实现

4.1 无冲突数据类型复制算法的设计与实现

第二章的需求分析中我们明白了冲突的场景，也了解了无冲突数据类型复制算法是规定了一种数据结构，而这个数据结构的关键是有一点就是形成对操作的唯一标识，从本质上来讲，是根据需求保存用户操作的意图[9]。

从数据结构的角度来看，我们将原来表示位置的线性表达信息，变成了一种树的结构来进行表达[10]。

4.1.1 Char 类

用户所操作的字符内容，字符位置被封装成了 Char 类。

```
type Position = Identifier[]
// 包装后的字符
class Char {
  position: Position
  value: string
  constructor(value: string, position: Position) {
    this.position = position
    this.value = value
  }
  compareWith(char: Char) {
    const pos1 = this.position
    const pos2 = char.position
    const minLength = Math.min(pos1.length, pos2.length)
    for (let i = 0; i < minLength; i++) {
      const comp = pos1[i].compareWith(pos2[i])
      if (comp !== 0) return comp
    }
    return pos1.length - pos2.length
  }
}
```

Char 具有两个属性 position 和 value，value 中存储的是单个字符，源于用户的本地操作或者远程操作，position 的类型为 Position，是 identifier 的数组[11]。

4.4.2 Identifier 类

```
/**
 * 作为 Char Position 的记录单位
 */
class Identifier {
  index: number
  userId: number

  constructor(index: number, userId: number) {
    this.index = index
    this.userId = userId
  }

  compareWith(identifier: Identifier) {
    if (this.index < identifier.index) {
      return -1
    } else if (this.index > identifier.index) {
      return 1
    } else {
      return this.userId - identifier.userId
    }
  }
}
export default Identifier
```

Identifier 类具有 index 和 userId 两个属性，前者是生成的，根据具体分配时树的空间剩余情况来决定，后者对用户来讲是唯一的，处理当 Identifier 比对时相同的特殊情况，处理更高用户优先级的用户来源[12]。

无论是 Char 还是 Identifier 类，都各自实现了自己的与同类的 compareWith 方法，方便上层进行比较[13]。

说完了存储用户操作的基本单位，接下来就是如何利用这样的数据结构的问题了。

4.4.3 CRDT 类

CRDT 类通过利用 Char 类进行用户信息的存储，在其属性中，有一个 struct 属性，struct: Char[] = []，是由 Char 的数据类型组成的一维数组，从 CRDT 类的使用者来看，关键是要实现两类调用，四种基本方法，即本地调用和远程调用，本地添加，本地删除，远程添加，远程删除这四个方法，从数据来源的角度来看，就是源于本地用户操作带来的操作和缘于远程或者说其他用户操作带来的操作[14]。

在介绍执行流程的细节之前，还需要把 Controller 类和 Editor 类介绍清楚，否则整体流程无法串联起来，因此会在介绍完 Controller 类和 Editor 类后，再介绍整体的代码执行流程，将其串联起来。

```
// 实践 CRDT 算法的文字管理器
class CRDT extends EventEmitter {
  userId: number
  struct: Char[] = []
  tree = new Tree()
… 省略其他 100 行代码
```

目前从 CRDT 类可以看出，用户的所有操作被存储到了 struct 中，而且是一个一维数组，而我们的编辑器是一个具有行列的二位空间，此外，所用 CodeMirror 当用户行为产生时，程序可拿到的数据如下。

```
interface EditorChange {
    from: Position;
    to: Position;
    text: string[];
    removed?: string[] | undefined;
    origin?: string | undefined;
}
interface Position {
    ch: number;
    line: number;
    sticky?: string | undefined;
}
```

从 CodeMirror 的类型文件中可知，表示位置的两个参数，ch，line，ch 表示是在哪一列，line 表示在哪一行，而我们的 Position 是一个一维上的树形结构，并不支持二维的表达方式，因此我们需要在两者之间进行适配，这个也主要是 Editor 类的工作内容。

Editor 类承担着 Codemirror 和 CRDT 类之间的适配，同步作用，具体的属性方面，Editor 类有着 modal 和 mde，前者是封装后的 CRDT 类的实例对象，后者是基于 CodeMirror 类封装后的实例对象，Editor 类通过 mde 监控用户的输入行为，进行转换处理，适配成 modal 层所要的，换句话说，适配成 CRDT 算法所需要的数据类型，另外 editor 类直接受 Controller 类管理，当远程用户产生输入行为时，传输的基本单位是操作的行为，如添加或者删除，加上 CRDT 的算法存储的 Char 类实例。因此当 modal 向 Editor 进行同步时，还需要去重新适配为 Editor 所能理解的格式。

这里再返回介绍下 CRDT 的空间分配方式。

```
generatePosBetween(
  posBefore: Position,
  posAfter: Position,
  newPos: Position = []
): Position {
  const identifierBefore = posBefore[0] || new Identifier(0, -1)
  const identifierAfter = posAfter[0] || new Identifier(WIDTH, -1) // WIDTH = 10

  if (identifierAfter.index - identifierBefore.index > 1) {
    // 范围分配足够
    const mid =
      identifierBefore.index +
      Math.floor((identifierAfter.index - identifierBefore.index) / 2)
    newPos.push(new Identifier(mid, this.userId))
    return newPos
  } else if (identifierAfter.index - identifierBefore.index === 1) {
    // 本层不够分配了，需要再开一层
    newPos.push(identifierBefore)
    return this.generatePosBetween(posBefore.slice(1), [], newPos)
  } else {
```

```
        // 需要去下一层比较
        newPos.push(identifierBefore)
        return this.generatePosBetween(
          posBefore.slice(1),
          posAfter.slice(1),
          newPos
        )
      }
    }
```

每一层的空间有限制的 WIDTH 大小，每次结合已有空间分配情况对半进行分配，以连续输入 1234561 为例子。

对应的逻辑空间就是一个这样的树形结构（存储逻辑上依然是数组，这是只是展示），结合上述算法来分析下这个加入过程

1 进入，generatePosBetween 生成的入参，posBefore 为[]，posAfter 为[]，newPosition 的算法递归产生结果的需要，这里就不提了，返回的结果为[[5]]

2 进入，generatePosBetween 生成的入参，posBefore 为[]，posAfter 为[5]，返回的结果为[7]

3 进入，generatePosBetween 生成的入参，posBefore 为[]，posAfter 为[7]，返回的结果为[8]

4 进入，generatePosBetween 生成的入参，posBefore 为[]，posAfter 为[8]，返回的结果为[9]

5 进入，generatePosBetween 生成的入参，posBefore 为[]，posAfter 为[9]，返回的结果为[9, 5]

注意这段判断代码 else if (identifierAfter.index - identifierBefore.index === 1)，全局规定了空间为 10，因此当 9 进入的时候，没有空间可以支持对半分配了，于是开辟了下一个空间，并在 0，10 之间取中间值 5 作为下一层的 index 值

6 进入，generatePosBetween 生成的入参，posBefore 为[]，posAfter 为[9, 5]，返回的结果为[9, 7]

1 进入，generatePosBetween 生成的入参，posBefore 为[]，posAfter 为[9, 7]，返回的结果为[9, 8]

因此，结合生成 char position 的逻辑，逆向还可以写出适配 codemirror 所需要的位

置。

```
findInsertIndex(char: Char) {
  let left = 0
  let right = this.modal.struct.length - 1
  let mid, compareNum

  if (
    this.modal.struct.length === 0 ||
    char.compareWith(this.modal.struct[left]) < 0
  ) {
    return left
  } else if (char.compareWith(this.modal.struct[right]) > 0) {
    return this.modal.struct.length
  }

  while (left + 1 < right) {
    mid = Math.floor(left + (right - left) / 2)
    compareNum = char.compareWith(this.modal.struct[mid])

    if (compareNum === 0) {
      return mid
    } else if (compareNum > 0) {
      left = mid
    } else {
      right = mid
    }
  }

  return char.compareWith(this.modal.struct[left]) === 0 ? left : right
}
```

从整体视角上来看，我们对用户的输入目前有两种格式，一个是 codemirror 自己持有的行列二维模式，这里称为模式 A，另一种是 CRDT 算法设计的基于 Char 和 Position 的数据模型，这里称为模式 B，当本地用户输入的时候，是从模式 A 适配到模式 B，当获取到远程用户输入的时候，是从模式 B 同步到模式 A，因此才有了上面所提的处理相互转换的代码[15]。

4.2 编辑器的设计与实现

最后来说一下处理 Editor 类和远程用户数据的 Controller 类，与整个多人编辑文档系统的设计模式。

4.2.1 Controller 类

```
interface EditorOptions {
  element?: HTMLElement
}
interface ControllerOptions extends EditorOptions {
  other?: any
}
export interface Operation {
  type: 'insert' | 'delete'
  char: Char
}
// 管理 Editor 与 Server 的通讯
class Controller {
  editor: Editor
  mockIO = new EventEmitter()
  constructor(controllerOptions: ControllerOptions) {
    const mde = new SimpleMDE({
      spellChecker: false,
      toolbar: false,
      shortcuts: {},
      element:
        controllerOptions.element ||
```

```
            (document.getElementById('editor') as HTMLElement),
        })
        this.editor = new Editor(mde, undefined)
        this.bindEmitterEvents()
    }
    bindEmitterEvents() {
        this.editor.on('remoteInsert', (char: Char) =>
            this.mockIO.emit('remoteOperation', {
                type: 'insert',
                char: char,
            })
        )
        this.editor.on('remoteDelete', (char: Char) =>
            this.mockIO.emit('remoteOperation', {
                type: 'delete',
                char: char,
            })
        )
    }
    applyOperation(operation: Operation) {
        if (operation.type === 'insert') {
            this.editor.remoteInsert(operation.char)
        } else if (operation.type === 'delete') {
            this.editor.remoteDelete(operation.char)
        }
    }
}
```

Controller 类就是作为远程服务器交互和本地 Editor 类的中间人，如当收到远程有其他用户发起了操作，Controller 类会通过 applyOperation 来执行相应的方法。mockIO 是为了本地开发方便提供的用于模拟网络 IO 的对象，而 editor 则是 Editor 类的实例。

4.2.3 Editor 类

Editor 在上文介绍 CRDT 类的时候已经阐述了 Editor 类负责衔接 modal 层和 CodeMirror 展示层的作用，还有数据格式适配的作用。此外，这里也是本地用户行为出发系统解析的起点。

```
bindChangeEvent() {
    this.mde.codemirror.on('change', (_: never, changeObj: EditorChange) => {
      // 只处理 local 用户的行为
      if (changeObj.origin === 'setValue') return
      if (changeObj.origin === 'insertText') return
      if (changeObj.origin === 'deleteText') return

      switch (changeObj.origin) {
        case 'redo':
        case 'undo':
          // this.processUndoRedo(changeObj);
          breakf
        case '+input':
          // case 'paste':
          this.processInsert(changeObj)
          break
        case '+delete':
          // case 'cut':
          this.processDelete(changeObj)
          break
        default:
          throw new Error('Unknown operation attempted in editor.')
      }
    })
  }
```

当本地用户进行编辑的时候，codemirror 会触发 change 事件，从这里可以根据事件中所包含的信息进行本地的增加和删除处理。

4.2.2 整体设计

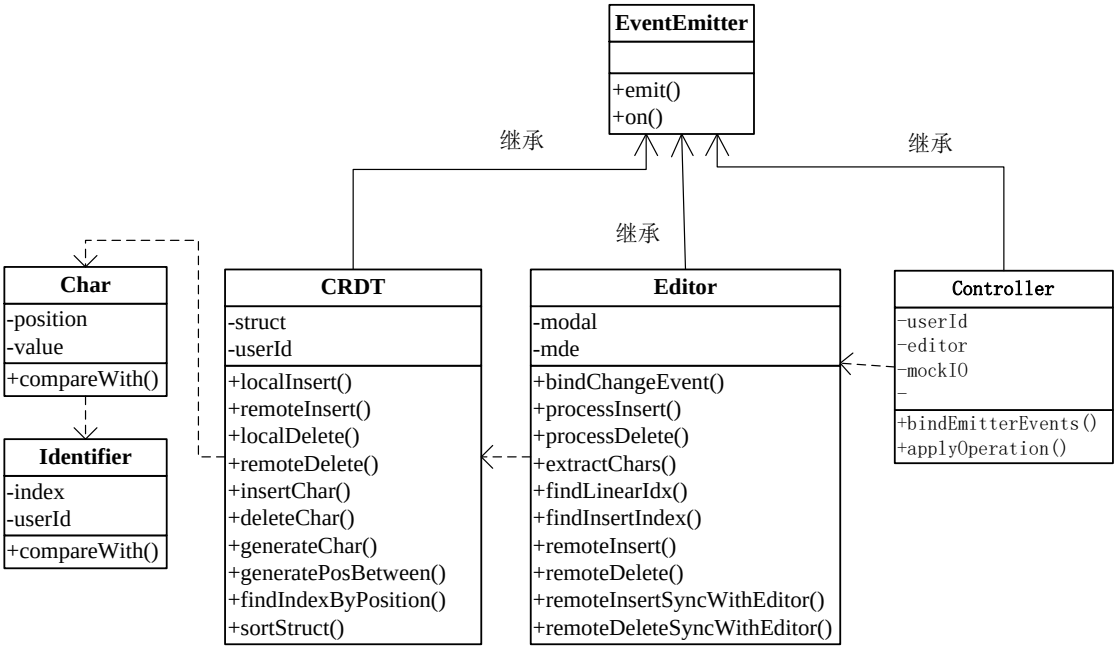

图 4-1 系统设计

当前上下文是只考虑三个类之间的关系，Controller 类、Editor 类，CRDT 类。

对于 Controller 来说，可见的只有 Editor 类，不可见 CRDT 类。对于 Editor 类来说，可见的只有 CRDT 类，而对于 CRDT 类来说，都不可见，那么当 CRDT 想要通知 Editor 类事情，或者 Editor 类想要通知 Controller 类事情，是怎样完成通讯的呢？

在这里引入了观察者模式，通讯的双方定义好通讯接口，利用 on 进行事件监听，利用 emit 来进行不同层的事件调用。这是一种降低耦合度的设计方法，当下层发生改变的时候，不需要改变引入的类，只需要保持原先的通讯接口不变，一样可以完成通讯过程，保证了整体设计的优化与灵活性。

Controller 类、Editor 类，CRDT 类，他们都继承了 EventEmitter 类，来完成发布与订阅能力。下面举个例子来展示这个过程。

以 CRDT 类执行了 localInsert 为例

```
localInsert(value: string, index: number) {
  const char = this.generateChar(value, index)
  this.insertChar(char)
  this.emit('remoteInsert', char)
}
```

可见对自己当前实例进行了 remoteInsert 事件的发布，外层 Editor 类存在下面方

法，并在 constructor 中进行了调用。

```
bindEmitterEvents() {
  this.modal.on('remoteInsert', this.emit.bind(this, 'remoteInsert'))
  this.modal.on('remoteDelete', this.emit.bind(this, 'remoteDelete'))
  this.bindChangeEvent()
}
```

modal 就是 crdt 的实例，Editor 类对这个实例进行了 remoteInsert 事件的订阅，并把这个事情直接抛给了自己再发布，当然，结合不同的场景这里可以再进行处理，在目前代码设计下，向服务器的传输过程中并不需要改变数据，因此当前采用了抛给自己再发布。

```
bindEmitterEvents() {
  this.editor.on('remoteInsert', (char: Char) =>
    this.mockIO.emit('remoteOperation', {
      type: 'insert',
      char: char,
    })
  )
  this.editor.on('remoteDelete', (char: Char) =>
    this.mockIO.emit('remoteOperation', {
      type: 'delete',
      char: char,
    })
  )
}
```

Controller 类再在外层进行了订阅处理，利用 mockIO 来模拟网络请求的过程。

4.3 本章小结

本章是整个论文的核心部分，分为两个大的部分来介绍系统设计，一、无冲突数据类型复制算法的具体设计实现，在这里具体介绍了与算法相关的三个实现类与相对位置生成的执行细节；二、编辑器的设计与实现，除了两个与编辑器相关的实现类外，在这里还从从整体上介绍了整个系统的架构和与设计模式的应用。

5 运行结果

从用户视角来看，两边的编辑器在逐个添加字符，删除字符的过程中，两边数据可以进行同步。并在下面给出了当前数据模型下面对应的树形逻辑结构。

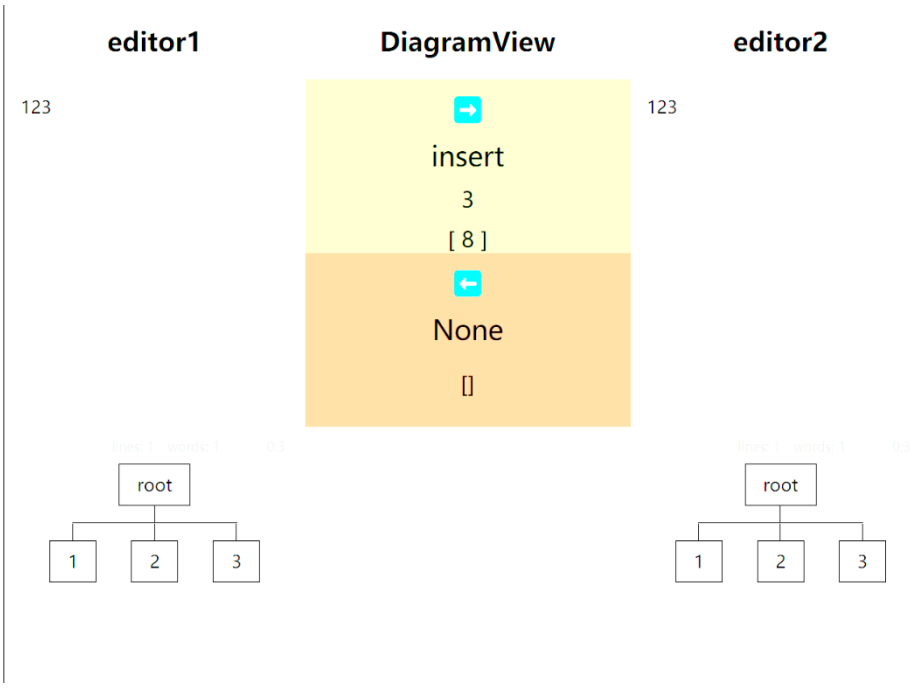

图 5-1 系统运行展示

```
      at CRDT.generateChar (src/crdt.ts:69:13)

---------------|---------|----------|---------|---------|-------------------
File           | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------|---------|----------|---------|---------|-------------------
All files      |   71.69 |     72.5 |   66.66 |   70.29 |
 char.ts       |     100 |      100 |     100 |     100 |
 crdt.ts       |   56.52 |    67.64 |      50 |   55.22 | ...-50,61-62,112-138
 identifier.ts |     100 |      100 |     100 |     100 |
 tree.ts       |     100 |      100 |     100 |     100 |
---------------|---------|----------|---------|---------|-------------------

Test Suites: 3 passed, 3 total
Tests:       28 passed, 28 total
Snapshots:   0 total
Time:        2.478 s, estimated 3 s
Ran all test suites.
```

图 5-2 系统测试结果

从研发的角度来看，除了和网页用户高度相关的直接交互部分，在整个软件的设计过程中，引入了许多单元测试。并取到了较好的测试成绩。

6 总结以及展望

回顾整个开发过程，我对在线多人编辑系统这样的相待高交互文档系统从一无所知到熟悉核心流程，经历了很长的时间，其间查阅了许多的论文、相关资料，整个设计的开始就充满了很多挑战，因为，无论是冲突处理算法还是 JavaScript 关于多个类的构建和相互合作，我都不太熟练，需要一点一点慢慢学，耐心看。

编辑器也是这个开发中所要处理的难点之一，得幸于良好的开源社区，codemirror 的存在减少了大量需要去考虑编辑输入和处理用户行为的代码，在浏览器的原生 input 之中，需要开发者自行去处理用户的输入行为，通过同时比对内容和键盘事件来进行行为的原子化包装，这一点利用了 codemirror 方便灵活的 API 接口得以快速实现。

在线编辑系统的实现过程中遇到了不少问题，比较阻塞的一点就是在多层的设计下，如何更加合理的迭代新功能呢，本身开发就我一人，限于时间考虑，引入自动化测试可以说是唯一的解法，不仅让我在引入新功能的时候不用担心老代码会出现问题，另外，有时我是先写测试在去设计和实现具体功能的，在这样的流程中，编码的目的性更强，相关功能点和特殊边界也考虑的更加全面。

本实践已经完成了多人在线协作所需的必要最小设计，限于对后端领域不甚了解，仅在本地利用 mockIO 这样的对象完成了两个用户同时编辑情况的模拟，完成了本地单个字符的输入，本地单个字符的删除，同步远程单个字符的输入，同步远程单个字符的删除这些能力，在基本原理知晓和核心流程完成的情况下，更多的功能也只是在模块上新增更多的操作处理和远程同步，部分能力或许还要优化下存储结构。

探索的有趣之处就是你会发现曾经未曾考虑过的风景，考虑到复杂度和个人能力，没有去使用业界在在线多人协作文档中常用的操作转化算法，而是使用了无冲突数据类型复制算法，而凑巧的是，在分布式系统设计中，无冲突数据类型复制算法是解决一致性问题的利器，业界也有许多成功的工具使用这一算法，如 Redis。无心的选择，在探索到后面，却给我开启了一扇通往分布式系统设计的大门。

最后再说下与后端交互的设计，C/S 架构下，服务是中心的，每个终端用户直接与服务器进行信息同步，而服务器再把大家的信息进行整合再同步，相较于操作转换算法，在 C/S 架构下，显然操作转换算法更具有优势，但这在用户量变大的时候，会对服务器提出更加具有挑战的性能要求，对于后端的架构设计会要求很高，其次，考虑中心服务器的地理位置，当中心服务器这个第三方中介距离大多数终端用户都比较远的时候，会带来比较大的操作延迟，给用户产生不好的使用体验，最后就是隐私性了，在 OT 算法

的设计中，中心服务器是清楚最终内容的，因为他要负责去进行操作的转换来满足文档的最终一致性，这就牺牲了隐私。再来看看使用 P2P 模式的优势，除了开始建立双发链接需要中介服务器的参与，其后的文档传输过程不再需要中心服务器参与，这极大降低了对后端服务器的性能依赖，并且没有第三方来把信息再传递，终端用户之间直接进行通讯，也缩短了网络传输的链路，提高了用户体验，这也提高了信息的隐秘性，保护了隐私。

总而言之，其中还有许多需要去再完善和改进的地方，我也从这次毕业设计中获益匪浅，不仅了解了在线多人编辑系统的实现细节，熟悉了 JavaScript 语言，熟悉了单元测试在项目开发中的运用，还初步感受了分布式系统设计下所面临的问题和对于一致性问题的解法，在本次设计中使用的树形位置标识和相关数据结构、算法的应用，也让我进一步深刻理解了数据结构和算法的意义，更加熟练的操作和使用树形结构来完成日常工作中的需求。

参考文献

[1] 申东伟. 企业文档编辑器的开发[J]. 信息通信, 2021, 20(4): 214-219.

[2] 王金峰, 彭禹, 王明等. 富文本编辑器的技术演进[J]. 计算机与网络, 2016, 42(22): 68-71.

[3] 徐尧. 2003 开发在线文档时，这个技术难点你解决了吗？[J]. 电子元器件与信息技术, 2020, 4(12): 156-157.

[4] 荣艳冬. 精彩 JavaScript 程序设计[J]. 信息安全与技术, 2015, 6(12): 86-88.

[5] 王丹, 朱思征, 高丽萍. 移动云平台下基于局部复制的结构性文档协同编辑冲突消解[J]. 电脑知识与技术, 2016, 12(14): 247-248.

[6] 翟成彤, 董海峰. 小型微型计算机系统[J]. 智能计算机与应用, 2019, 9(2): 182-186.

[7] 何发智, 吕晓, 蔡维纬. 支持操作意图一致性的实时协同编辑算法综述[J]. 现代信息科技, 2020, 4(18): 125-127.

[8] 王恩生. 系统软件开发过程中的软件工程技术[J]. 计算机与网络, 2020, 46(9): 68-71.

[9] 高丽萍, 陈庆奎, 姚一成. 持团队分工的实时协同一致性维护技术研究[J]. 山西科技, 2020, 35(6): 45-47.

[10] 邵斌, 卢暾, 顾宁. 实时协同中的一致性维护关键技术[J]. 计算机工程, 2020, 16(3): 3-4.

[11] 贾小云, 杜晓旭. 协同组编辑中基于地址空间转换的一致性维护方法[J]. 计算机应用与软件, 2020, 37(8): 33-38.

[12] 杨友东, 周勋, 朱根兴. 集中式同步协同设计系统关键技术研究[J]. 电脑与信息技术, 2019, 27(4): 25-28.

[13] 邵斌. 高效的操作转换一致性维护方法研究[J]. 青岛大学学报(工程技术版), 2017, 32(4): 114-119.

[14] 黄秀丽, 陈志. 基于 JSON 的异构 Web 平台的设计与实现[J]. 计算机技术与发展, 2021, 31(3): 120-125.

[15] 范志. 实时协同编辑系统中并发控制算法的研究[J]. 指挥信息系统与技术, 2020, 11(6): 96-100.

致谢

本论文是在指导老师王春的帮助下完成的，时间过得可真快啊，一转眼四年的大学生活就要结束了，在这快四年的学习中，我身边的很多老师和同学都给了我很大的帮助，在此我对他们表示衷心的感谢。疫情的大背景下，尽管许多工作都是在线上完成的，但在王春老师的认证负责下，依然达到了保质保量的完成了毕业论文和毕业设计相关的工作，这个过程中，老师始终及时的解答了许多我的困惑和问题，短短的几个月时间，无论是对相关问题的理解，还是相关编程工具的使用，其熟练度和理解的深度都得到了显著的提升。同时，也感谢学习相伴的同学们，大家一起聊着进度，一起解决格式方面的困惑和画图相关工具的困难，与同学之间的记忆和友谊，也是我这最后大学阶段中宝贵的记忆。

最后，我还要感谢我的母校，淮北师范大学，我的大学时光中，很多的时间都在实验室度过，感谢母校提供的宝贵学习环境，这些宝贵的经历，将是伴随我走向社会的坚实后盾。

作者：胡晨曦

2022 年 4 月 11 日