清 华 大 学

计算机科学与技术系

计算机专业实践

课题名称：多人协作云脑图

姓　　名 李成江　　学号 2013011305　班号 计 32

指导老师 徐昆　　辅导老师 蓝东飞

成绩_____

二零一六年八月

目　　录

1. 多人协作云脑图背景及意义

1.1 研究目标

1.1.1 项目背景

本项目的提出原因是为腾讯公司提供一款可供多人协作编辑的 web 端脑图，以供头脑风暴或会议使用。

1.1.2 相关研究现状

开始项目之前，我们对市面上已有的脑图编辑器进行了调研，结果和分析如下：

1. 在线多功能图表绘制软件：以 ProcessOn 为例，这类编辑器的特点是除了脑图还能编辑其它类型的图表，但相对的只有基本的编辑功能。

(a) 优点：

- 除了脑图还能编辑其它图表；
- 支持多人在线协作；
- 在线编辑，支持多平台，不需下载客户端；
- 支持多种格式导出。

(b) 缺点：

- 只有基本的编辑功能。

2. 在线脑图绘制软件：以百度脑图为例，这类编辑器的特点是功能丰富。

(a) 优点：

- 绘制脑图的功能丰富；
- 界面优美，用户体验好；
- 在线编辑，支持多平台，不需下载客户端；
- 支持多种格式导出。

(b) 缺点：

- 不支持多人协作。

3. 本地脑图绘制软件：以 XMind 为例，这类编辑器的特点是功能丰富，但往往要收费。

(a) 优点：

- 绘制脑图的功能丰富；
- 进阶功能多（如作为幻灯片播放）；
- 支持多种格式导出。

(b) 缺点：

- 需要下载客户端使用，不支持多平台;
- 收费。

除以上列举优缺点之外，以上所列脑图均没有和已有的敏捷开发平台对接的功能。

1.1.3 实现目标

在 TAPD 上实现在线云脑图，开放多人协作，并支持一键转为 TAPD wiki 和需求。便于头脑风暴会议、工作讨论的记录与转存，开拓思维，沉淀创意。

具体分为以下四点：

1. 脑图

作为一款脑图编辑器，我们的脑图将拥有基本的脑图编辑功能。

2. 云端

相对于需要在本地下载客户端的脑图编辑器，我们的脑图只需在网页上即可编辑。

3. 多人协作

支持多人同时在线编辑同一脑图。

4. 面向敏捷开发

与 TAPD 对接，支持将脑图与 TAPD 的需求/wiki 一键转化。

1.1.4 可行性分析

通过对市场上已有产品的分析，能确定一个功能强大、支持多人协作、能够和敏捷开发平台对接的脑图编辑器是能够实现的。由于博采众长并发挥自身特点，我们的产品将能在市场中有自己的一席之地。

1.2 内容

1. 功能需求：
   (a) Web 页面上实现在线云脑图的创建、保存；
   (b) 支持多人协作编辑与分享；
   (c) 支持一键转为 TAPD wiki 和需求。
2. 相关技术
   (a) 基于 H5 图形库的前端可视化操作；
   (b) 基于竞争机制的多人图形操作；
   (c) 基于 TAPD openAPI 的数据同步。

2. 实验方案

2.1 项目设计及实施方案

2.1.1 技术选型

1. 前端：

(a) 采用 AngularJS 框架

ⅰ. 基于 MVC 架构，模块划分清晰，便于分工合作

ⅱ. 监听 DOM 事件的方式直观易用

ⅲ. 自动根据 M（model）更新 V（view）

(b)使用 HTML5 SVG 绘制节点、树边等图像元素

ⅰ. 编程方式作图，精确且易修改

ⅱ. 矢量图，缩放后不失真

ⅲ. 作为 HTML 元素，便于和 AngularJS 提供的 ng-repeat、ng-click 等 DOM 相关操作方式结合

2. 后端

(a) Node.js

ⅰ. 用事件调度、异步处理的机制实现高并发

ⅱ. 非传统的多线程机制，避免了多线程下的冒险

(b) MongoDB

3. 通讯

(a) Socket.IO

ⅰ. 基于 WebSocket，提供前后端间的 socket 连接

ⅱ. 简单易用的 API

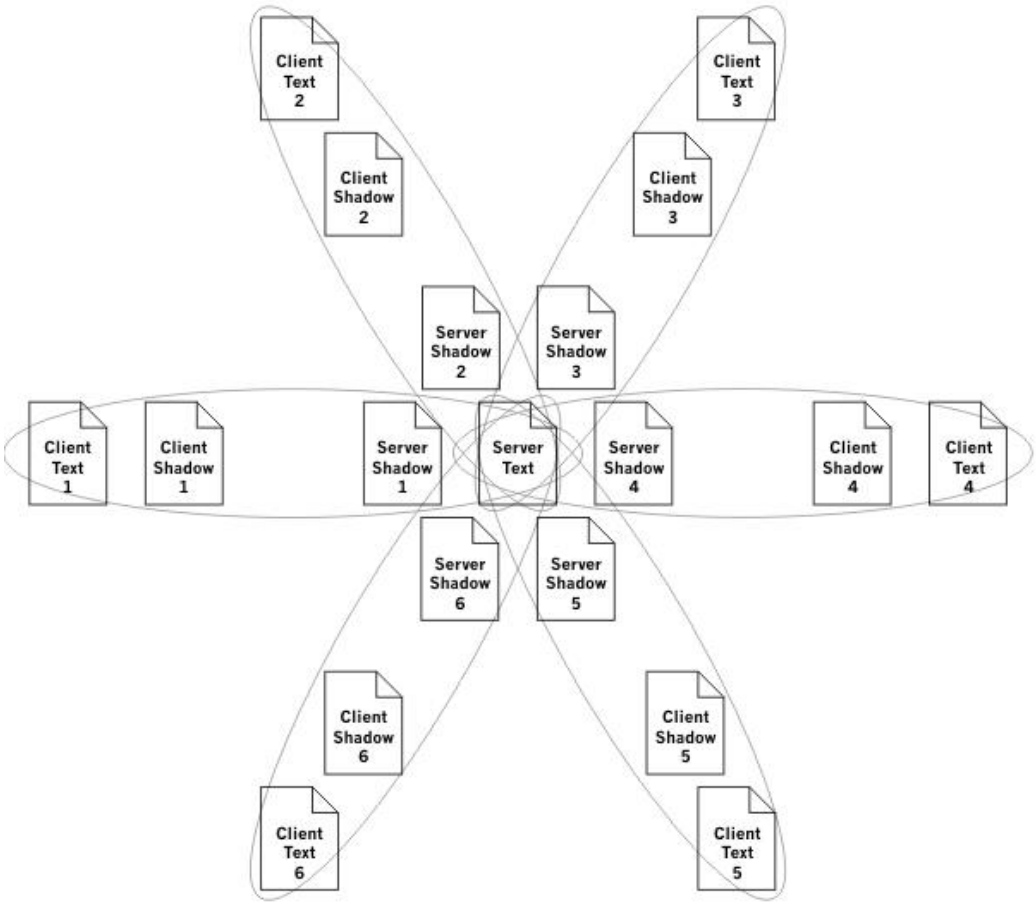
(b) Differential Synchronization

ⅰ. 实现多人实时协同编辑的核心算法

2.1.2 协同编辑算法

为了支持协同编辑，我们采用的同步算法叫做 DIfferential Synchronization，以下简称 DiffSync，该算法是由 Neil Fraser 于 2009 年提出来的。

考虑这么一个应用场景：有六个人要同时编辑一分文档，这份文档存在于一个服务器上。下图是采用 DiffSync 时的拓扑结构，中心是服务器，六个角代表六个客户端。



基本概念：

为了阐明算法的基本思想， 需要先介绍 4 个概念：

Server Text

1. 存在于服务器，只有唯一的一份
2. 是用户们最终希望编辑的目标

Client Text

1. 存在于客户端，每个客户端有独立的一份
2. 可以看作是 Server Text 在每个客户端的镜像
3. 每个客户端编辑文本时，是首先直接作用在 Client Text 上的，之后才由同步算法把编辑操作传递到 Server Text 上

Client Shadow

1. 每个客户端都有独立的一份
2. 逻辑上，是各个客户端上 Client Text 的快照
3. 具体来讲，是客户端向服务器发起同步请求时的快照

Server Shadow

1. 存在于服务端上，为每个客户端维护一份
2. Server Shadow 和 Client Shadow 对应
3. 逻辑上，Server Shadow 和 Client Shadow 是同一个东西，区别仅仅在于前者存在于服务端，后者存在于客户端
4. Server Shadow 和 Client Shadow 通过传输 diff、打 patch 的方式保持内容一致

同步流程：

现在假设有一个客户端对它的 Client Text 进行了修改，那么 DiffSync 是如何实现同步的呢？

接下来就来介绍算法同步的步骤：

1. 客户端 i 编辑了和它对应的 Client Text[i]，启动同步流程。

   注意：此时 Client Shadow[i]和 Server Shadow[i]的内容是一样的，毕竟它们逻辑上是同一个东西。
2. 客户端首先对比 Client Text[i]和 Client Shadow[i]，产生一个 diff，不妨命名为 DiffA。

   注意：当根据 DiffA 向 Client Shadow[i]打 patch 时，Client Shadow[i]将变得和生成 diff 时的 Client Text[i]内容相同。
3. 这个 DiffA 相当于一个编辑操作。
4. 客户端根据 DiffA 向 Client Shadow[i]打 patch，并把 DiffA 发送给服务端。
5. 服务端接受到 DiffA 后，根据 DiffA 向 Server Shadow[i]打 patch。
6. 被打 patch 后，Server Shadow[i]和 Cllient Shadow[i]就又一样了。
7. 把 DiffA 视作一个编辑操作，在这步里我们根据 DiffA 向 Server Text 打 patch。

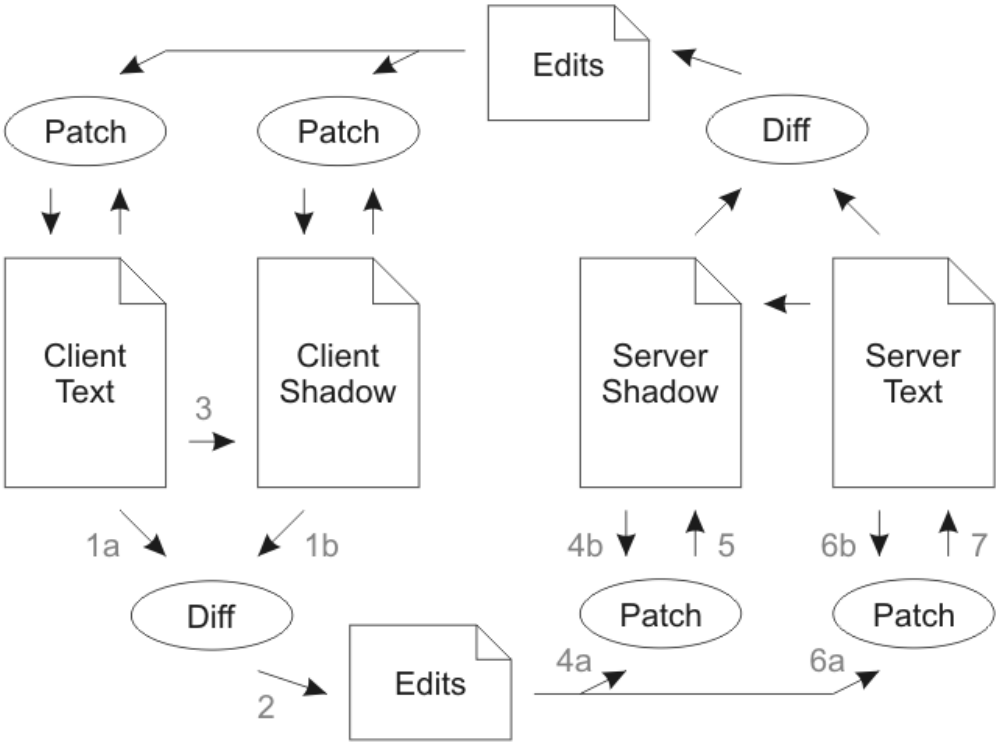
   注意，这一步打 patch 可能会失败，具体解决方法见后。
8. Server Text 被打 patch 后得到了更新。
9. 之后，服务端对比 Server Text 和 Server Shadow[i]，获得一个 DiffB。

   注意：服务端可能会在接受并处理来自多个客户端的 DiffA 后才进入这一步。
10. 根据 DiffB 向 Server Shadow[i]打 patch，如此一来 Server Shadow[i]就和 Server Text 内容一样了。
11. 将 DiffB 反馈给刚刚发起同步流程的客户端，同时服务端通知其他客户端有更新。

    细节：其他客户端将根据维护的版本号等机制检查是否需要和服务端同步。
12. 客户端接收到 DiffB 后，根据 DiffB 向 Client Shadow[i]打 patch。

    注意：Client Shadow[i]、Server Shadow[i]再次一致。
13. 客户端尝试根据 DiffB 向 Client Text[i]打 patch。

    注意：这一步打 patch 可能会失败，具体解决方法见后。



参考实现：

GitHub 上有名为 Jan Monschke 的开发者提供了上述算法的开源参考实现（MIT License）。尽管这是我们能找到的较好的实现，它还是存在很多问题。

首先是功能上的缺陷：

1. 无关闭连接的功能：导致存在严重的内存泄漏；无法安全地实现删除文件。
2. 无访问控制、安全认证：可轻易地获取数据库的任意内容。
3. 任何连接均可读可写：不能安全地实现只读的实时分享。

其次在正确性上，还有更为致命的问题：该算法没有考虑前述步骤 7、步骤 13 可能打 patch 失败的问题，导致该实现及其容易崩溃。

改进实现：

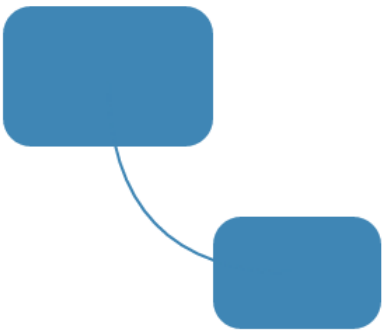
我们 fork 了该参考实现，然后对其进行了大幅修改，增加上述缺失的功能，同时我们修复了原参考实现中致命的实现不当问题。

2.1.3 复杂图像绘制

为了绘制脑图，前端使用 SVG，如下述代码：

```
<svg>
    <rect x="0" y="0" width="150" height="100"></rect>
    <rect x="150" y="150" width="120" height="80"></rect>
    <path d="M 75 50 Q 75 190 210 190"></path>
</svg>
```

绘制结果如下图：



但是用这种方法绘制脑图需要列举许多重复的元素。虽然可以生成对应代码嵌入 HTML 中，但这种方法并不够好。最终我们使用了 AngularJS 中的 ng-repeat 来解决这个问题，最终代码形式如下：

```
angular.module("myApp", []).controller("myCtrl", function($scope) {
    $scope.nodes = [{x: 0, y: 0, width: 150, height: 100},
                    {x: 150, y: 150, width: 120, height: 80}];
    $scope.edges= [{x1: 75, y1: 50, x2: 210, y2: 190}];
});
```

```
<svg ng-app="myApp" ng-controller="myController">
    <g ng-repeat="node in nodes"
        <rect ng-attr-x="{{node.x}}" ng-attr-y="{{node.y}}"
            ng-attr-width="{{node.width}}" ng-attr-height="{{node.height}}">
        </rect>
    </g>
    <g ng-repeat="edge in edges">
        <path ng-attr-d="M {{edge.x1}} {{edge.y1}} Q {{edge.x1}} {{edge.y2}}
                {{edge.x2}} {{edge.y2}}"></path>
    </g>
</svg>
```

实际实现时，上述代码中的$scope.nodes、$scope.edges 不是手写的，而是根据脑图 model 经过排版算法得到 js 中的具体数据并传给 HTML，从而顺利显示脑图。

2.1.4 用户身份认证

由于项目目标是接入 TAPD 应用，应用的打开方式为：点击应用图标，在浏览器新标签页访问一个 URL，URL 中的 GET 参数部分包含一个由用户名和 workspace_id 加密而成的密文串（由 TAPD 提供）。服务端接到访问之后，对传入参数进行解码，获知用户信息完成认证。

所以项目的用户认证方式不同与传统应用的“登录”操作。

URL 组成：

http://HOST_NAME:PORT/projectevent?params=PARAMS

说明：

1. HOST_NAME：主机名
2. PORT：项目占用的端口号
3. PARAMS ：TAPD 生成的包含当前用户名及 workspace_id 的密文串（详情咨询 AnyeChen）

Example:

http://10.125.48.28/projectevent?params=\\slSlqZ2miKBTrsHJoc\%2FMUpKFp6PXo6aoxZWZkpyUW5tZY2Bna2WemmJa3w%3D%3D

2.1.5 数据存储方式

脑图的 JSON 格式 脑图模型为一棵由多个节点构建出来的树。

基本节点模版：

```
{
  "content": {
    "text": "" //  节点文本内容
  },
  "style": { //节点在脑图画布的坐标
    "X": 0,
    "Y": 0
  },
  "extra": { //  拓展节点属性的字段
  },
  "tags": [    //  节点 tag（如果有）
  ],
  "children": [], //  该节点的子节点存放
  "id": 0 //  每个节点唯一的 id（根节点为 0）
}
```

由此构成了前后端统一使用的脑图模型。

实际实现时，为了支持更多功能，采用的模版比这个略复杂一些。

存储数据的索引名 数据库使用 MongoDB，可以直接存储 JSON 格式的数据。数据格式为：

```
var ModelSchema = new mongoose.Schema({
    // 脑图模型名字（命名规则见下文）
    name: {type: String},
    // 脑图的模型
    model: Object,
    // 加快分类查询的索引，为 workspace_id  或  user_name
    workspace: {type: String}
});
```

脑图模型命名规则

WORKSPACE + '@' + TYPE + '@' + MODEL_NAME

脑图模型主要分三类：

1. 个人脑图：user_name(English) + '@' + 'USER' + '@' + modelId(model_name);
2. 迭代：workspace_id + '@' + 'ITER' + '@' + modelId(model_name);
3. Wikis：workspace_id + '@' + 'WIKI' + '@' + 'Live';

2.1.6 编辑功能实现

撤销、重做功能：

问题描述：如何实现编辑时的撤销、重做功能。

解决方案：

1. 在前端为每个用户维护两个历史操作的栈：undo 栈、redo 栈。

2. 当用户进行操作 op 时，将操作的逆操作 inv(op)压入栈 undo 栈。
3. 当用户撤销时，undo 栈弹栈得到 inv(op)操作，执行 inv(op)，同时将弹出的元素的逆操作，即 inv(inv(op))，也就是 op，压入 redo 栈。
4. redo 功能的实现类似。

实现效果：每个用户可以独立的撤销自己的操作，不干扰其他用户的使用。

节点文字编辑框：

问题描述：我们希望用户能够编辑显示于节点上的文字。这段文字在是由 SVG 内的<text>元素负责绘制的，但是<text>不支持编辑。

解决方案：

如果试图通过在 SVG 内将原本的<text>替换成<textarea>或<input>来达到编辑功能的话，会发现行不通，因为<textarea>、<input>等这些常规的 HTML 元素是不能放入 SVG 元素中的。

因此，我们采取的方案是在<body>中放置一个隐藏的<textarea>元素，当用户点击某个节点的文字时，便将该隐藏的<textarea>移动到该段文字的正上方、然后取消隐藏，从而在文字的正上方为用户提供一个编辑框。采用了类似实现方法的还有备注框等功能。

缩放画布：

问题描述：如何实现缩放功能。

解决方案：

1. 在数据库中维护节点的逻辑坐标，前端显示时，维护缩放比例 P 和屏幕中心对应的逻辑坐标(x_0, y_0)、画布长宽 H、W。
2. (x, y)显示于像素点(p_x, p_y)，且始终存在着这样的对应关系。

$$(x - x_0) * P = (p_x - W / 2)$$
$$(y - y_0) * P = (p_y - H / 2)$$

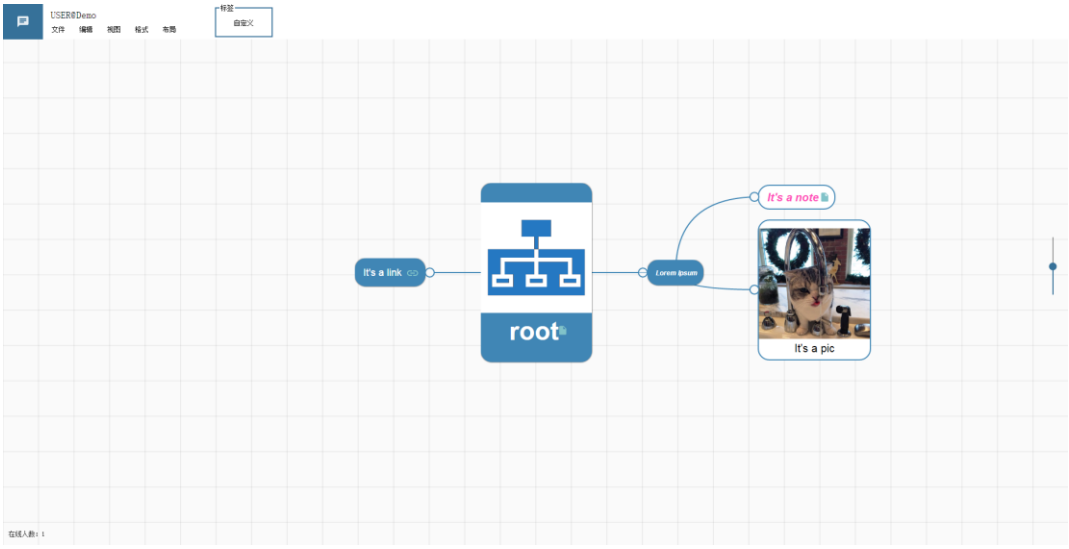
3. 缩放时，维护缩放中心的对应关系不变，改变 P 值和(x_0, y_0)即可。
4. 鼠标位置为(x_m, y_m)，新的 P 值为 newP，则 x_0、y_0 需要经过重新计算：

$$x_0 = (x_0 - x_m) * P / newP + x_m$$
$$y_0 = (y_0 - y_m) * P / newP + y_m$$

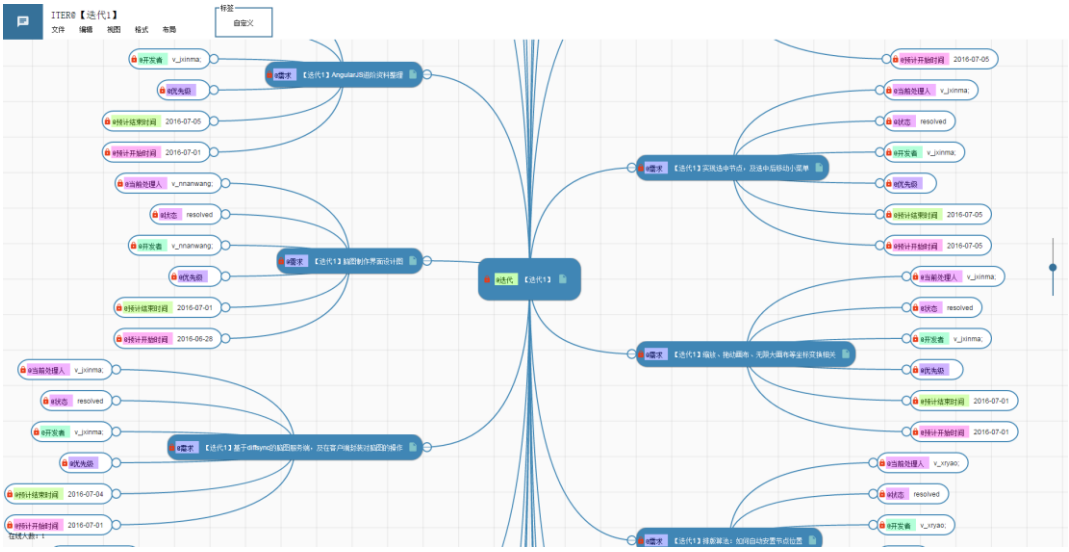
3. 实验结果

完成一个支持多人协作的脑图应用，支持与敏捷开发平台 TAPD 交互，部署于腾讯内部服务器。

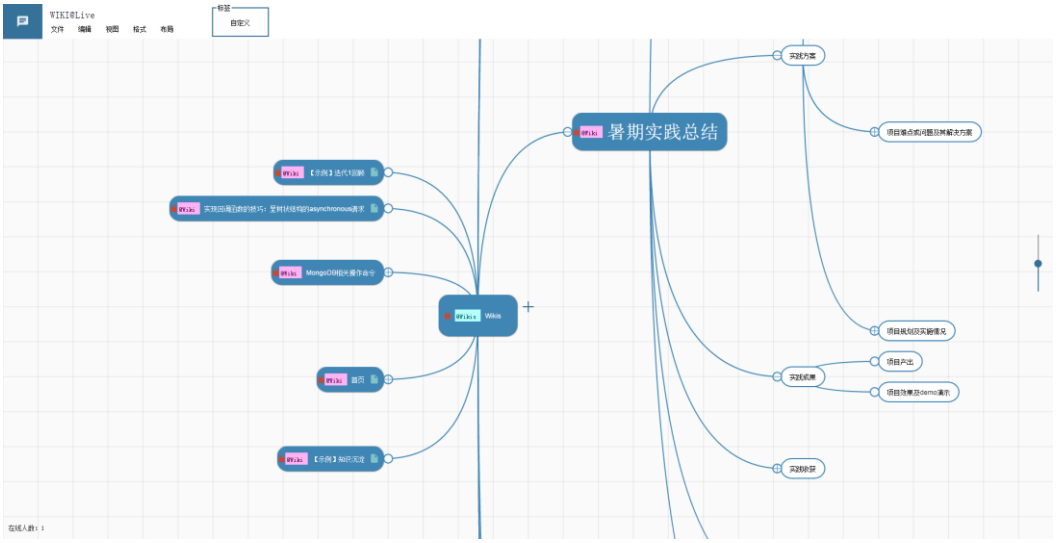
基本功能界面展示：



与迭代（需求）交互界面：



与 Wiki 交互（复杂文档编辑）界面：



4. 主要难点和解决方案

4.1 协同编辑算法

问题描述：如本文档前述，同步算法的参考实现在一处关键之处实现不当，没有考虑到向 Server / Client Text 打 patch 即使正常情况下也可能会失败的情形，因此及其容易崩溃。

解决方案：

关于前述同步算法的步骤 7、步骤 13 中打 patch 失败的解决方法，我们首先需要知道打 patch 之所以失败的根本原因是：生成 Diff 时，假定的打 Patch 目标是 Server / Client Shadow，因此，这个 Diff 不一定适用于 Server / Client Text。

举个例子，如果客户端 A 删除了节点 x，而客户端 B 移动了节点 x，那么服务端如果先接收到客户端 A 的 diff、再接收到客户端 B 的 diff，那么使用客户端的 diff 向 Server Text 打 patch 可能成功，用客户端 B 的 diff 向 Server Text 打 patch 时却会因为节点 x 已经删除而打 patch 失败。

步骤 7，在服务端向 Server Text 打 Patch 失败时，解决方法是：

1. 预先判断向 Server Text 打 Patch 能否成功。
2. 若不能，则直接跳过向 Server Text 打 Patch 的操作。因为 Server Shadow 随后在步骤 8-9 会被更新到和 Server Text 一致，因此可以这样做。

步骤 13，在客户端向 Client Text 打 Patch 失败时，解决方法：

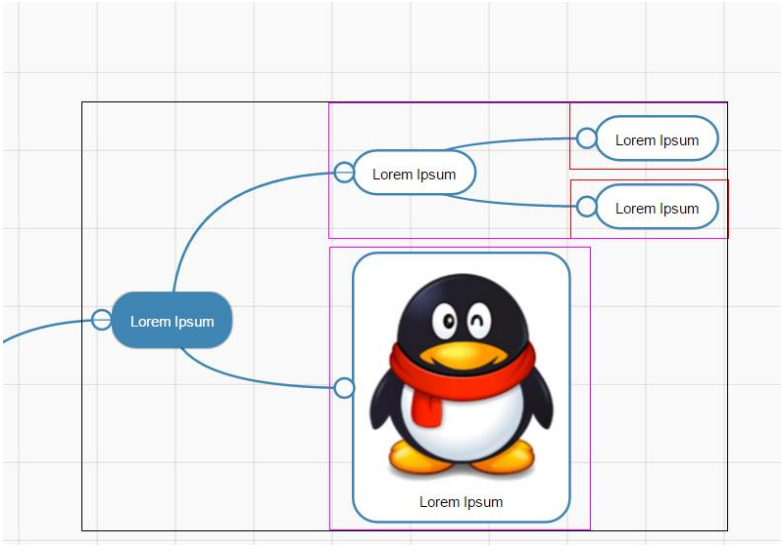
1. 步骤 11 里，先根据 Diff 更新 Client Shadow，再尝试根据 Diff 更新 Client Text 。
2. 更新 Client Text 时，预先判断向 Client Text 打 Patch 能否成功。
3. 若不能，则将 Client Text 回滚到和 Client Shadow 的内容相同。此时 Client Shadow 的内容是刚刚更新的，所以损失较小。

4.2 节点位置规划

问题描述：如何把节点美观、不重叠地绘制在网页上

解决方案：

1. 从根开始，后序遍历整棵树。
2. 对于每一个节点，先调整其子节点位置，使子节点的中心与自身中心对齐。
3. 根据子节点返回的高度值，计算以自身为根的子树所占据的高度。



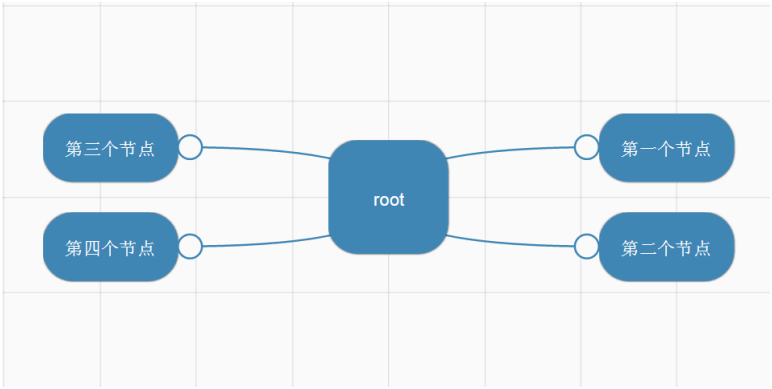
4.3 放置新增节点

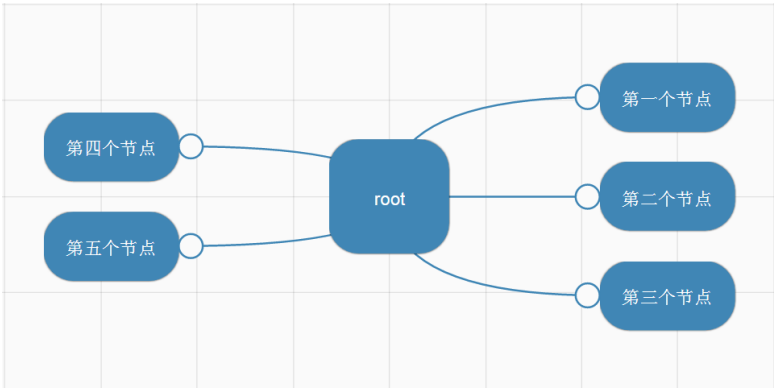
问题描述：当用户要在某个节点下新增加一个节点时，如何合适地设置新添加节点的初始坐标、并适当调整周围节点的位置。

解决方案：

1. 首先新节点的父节点是否为根节点，若不是，则添加到父节点下一级即可，同时，平移所有子节点，使子节点的中心与父节点对齐。

2. 若父节点为根节点，则向根节点左侧最下方添加节点，若左侧节点个数大于右侧，则把左侧最上方的节点添加到右侧最下方。之后再平移左右侧节点，使子节点的中心与父节点对齐。





4.4 TAPD Open API 使用问题

问题描述：TAPD 开放平台未提供 JavaScript 语言的 API SDK。

解决方案：参考提供的 SDK，编写 JavaScript 语言的 SDK。方便项目拓展使用到更多的 API。

问题描述：update_story 请求参数中列有需求规模 size，但发 POST 请求时添加 size 会请求失败。

解决方案：经与 AnyeChen 沟通，发现这是接口参数的一个问题，更改子需求 size 会自动更改夫需求 size，为了避免大量计算，所以之前这个参数并不提供设定。之后开放，但限制调用频率。为了不影响产品正常使用，暂未在脑图模版中添加 size 项。

问题描述：网页上的 Tapd Wiki 存在层次结构，但提供的 API 参数中不含能体现这种关系的字段。

解决方案：经与 AnyeChen 沟通，发现这部分接口的参数暂未开放。所以在 Wiki 脑图中我

们不呈现层次关系，仅将所有 Wiki 作为更节点的一级子节点呈现。

问题描述：get_wikis 无论如何指定请求参数，都会将所有的 Wiki 返回，包括在网页上已删除的。
解决方案：经与 AnyeChen 沟通，发现这部分接口的参数存在 Bug。在进一步了解如何通过特定字段区分 Wiki 状态后，后端将获得的所有 Wiki 先做筛选，再供下一环节使用。

问题描述：与 Wiki 相关的只有 get_wikis 和 add_wiki 两个接口，并不足以满足我们对 Wiki 同步编辑的操作需求。
解决方案：经与 AnyeChen 沟通，发现 update_wiki 接口暂未开放。经测试后发现，添加 Wiki 时所给的 name 为 Wiki 在 TAPD 上唯一性标志，两次 add 同一 name 的 Wiki 将产生内容覆盖的编辑效果。故同步编辑时，直接调用 add_wiki 接口，不允许用户修改已有 Wiki 的 name。

5. 课题产出

5.1 基本脑图功能

我们的脑图包含以下基本编辑功能：

1. 文档

(a) TAPD

- 迭代可创建、打开、关闭、另存为。
- 未保存到 TAPD 的迭代可重命名、删除。
- wiki 可打开、关闭。

(b) 个人脑图

- 个人脑图可创建、打开、关闭、另存为、重命名、删除。

2. 脑图

(a) 基本功能

- 节点可增添、删除、修改内容。
- 节点可拖动修改位置。若拖动到其它节点上，则能修改父子关系。
- 节点可展开、折叠。
- 节点可拖动，可一键重新排版。
- 脑图操作支持撤销和重做。

(b) 进阶功能

- 节点文字的颜色、大小可修改，并可设置粗体与斜体。
- 节点位置改变时有滑动的动画效果。
- 节点上可添加图片、超链接、备注、标签。
- 节点可复制、剪切、粘贴。
- 脑图可导出为 svg、png。
- 脑图可分享只读链接，可实时看到当前修改结果。

5.2 多人协作功能

支持多人同时编辑同一脑图，能够处理多人操作中的冲突。

5.3 敏捷开发功能

支持将脑图和 TAPD 的相互转化，具体方式如下：

1. 需求

   - 一个迭代对应一个脑图文档。

   - 一个需求对应一个节点，此节点需包含“@需求”标签。

   - 需求标题为节点内容，需求内容为节点备注（HTML 格式），需求内容需在 TAPD 页面编辑。

   - 一个迭代状态（开始、结束日期）或一个需求状态（当前处理人、开发者、开始日期、结束日期、优先级、状态）对应一个节点，此节点需包含一个以“@”开头的标签（具体内容由状态内容指定）。

   - 含锁标记的节点不可删除。

2. wiki

   - wiki 对应一个脑图文档。

   - 一个 wiki 文档对应一个节点，此节点需包含“@Wiki”标签。

   - 一个 wiki 文档的标题对应一个节点，此节点需包含在 wiki 文档对应节点的子树中。

   - wiki 文档的内容对应一个节点的备注，备注使用 markdown 格式书写。

5.4 复杂文档编辑功能

在与 wiki 的对接工作中发现我们的产品可以用来进行文档的书写，即通过节点的层级关系与顺序关系来生成一篇 markdown 文章。

因为项目完成度较高，定位是成为 TAPD 内部应用，所以之后有可能被测试后作为正式应用而使用。

6. 总结、收获和展望

技术方面，熟悉了后端 JavaScript 编程、MongoDB 数据库的使用，前端 AngularJS 框架及其使用；能力方面学会了灵活使用 Google，Github，Npm package 解决遇到的问题；更为重要的是较为全面的学习到了一个 Web 项目完整的开发流程；

产品方面，因为我们项目的处理多人协作方面使用的是 Websocket 和 Diffsync，之后对于再次实现多人协作的应用，我们仍然可以使用这套结构。学到的知识和获得的经验希望能在以后的开发中使用出来，继续学习。

目前我们项目在多人协作中，区分用户正在编辑哪个区域的体现并不具体，我们只能通过另一用户的修改更新来看出这个节点正在被修改。所以之后如果继续开发的话，应该会着重在多用户区分上。目前想到的一种方案，是用不同颜色来区分不同用户（每个用户的颜色在该文档内唯一），用户在选中、编辑、拖动节点时，节点能变成对应的颜色，并在功能上不允许其他用户对其修改。另外，在用户权限方面，可以考虑实现 Leader 权限，运行其修改所有的脑图，而员工只允许处理自己项目的脑图，并在进行重要操作（同步、删除等）时必须先向 Leader 发请求。

很希望我们组的产品之后还能有人来维护拓展，并在未来的某一天出现在腾讯的上线应用中。

7.体会和感想

我的家在云南临沧，作为一个南方人，确实很有可能在毕业后选择在一个南方的城市生活和发展。这次实践让我有机会到深圳，感受了与家相似的气候与饮食，让我挺喜欢这个城市的。离香港很近，让我们在周末也有机会去开阔视野休闲娱乐。第二周的周末，我们一起去了香港的海洋公园。游玩的同时，也让我们了解了彼此，在之后的工作中沟通更加自如。

在腾讯，我提前接触到毕业后的工作形式，感受到项目刚开始时的兴奋与新鲜，也在项目中后期体会到每日一成不变的乏味。工作中会遇到自己未接触领域的问题，看 Google 学习之后，再向同事请教，总能得到较为满意的方案。项目做成会有喜悦，但也会让自己思考这是不是自己所期望的工作方式。两个月的实习很有收获，很感谢组员之间互帮互助的学习，也很感谢导师与其他同事给予的帮助。

我觉得实践的组织很好，由于学校承担了交通与住宿费用，所以在很大程度上减轻了我们在深圳生活的压力。在建议方面，唯一的一点，就是希望能参与到实际腾讯项目的开发中。因为目前的形式，虽然我们会与腾讯同事联系，但大部分时间还是和同学在一块开发小组的项目，对实际工作中会遇到的问题、交流沟通、项目压力的感觉会少很多。比如最终的展示会，是我收获最多的一次会议，因为真的听到了来自上层不同的声音，真的会有人和你 argue，自己也在这个过程中学到了如何面对别人挑战，而不是你好我好大家好，一味的求和谐。而这样的会议在工作中是经常遇到的。我觉得多面对这样的挑战，会对我们的成长更有帮助。所以希望以后的实践能考虑多与腾讯实际在进行的工作结合，让我们多和员工接触。

参考文献

Neil Fraser. January 2009. *Differential Synchronization*. https://neil.fraser.name/writing/sync/