南京大学

研 究 生 毕 业 论 文

（申请工程硕士学位）

论文题目	基于React的超级账号教学支持系统前端的设计与实现
作者姓名	邬文怀
学科、专业名称	工程硕士(软件工程方向)
研究方向	软件工程
指导教师	任桐炜 副教授

2018 年 04 月 20 日

学　　号：**MF1632080**

论文答辩日期：　　年　月　日

指 导 教 师：　　（签字）

基于 React 的超级账号教学支持系统前端的设计与实现

作　　者：　　鄔文怀

指导教师：　　任桐炜 副教授

南京大学研究生毕业论文

（申请工程硕士学位）

南京大学软件学院

2018 年 04 月

The Design and Implementation of the front-end of SuperID Teaching Support System Based on React

Wu, Wenhuai

Submitted in partial fulfillment of the requirements for the degree of Master of Engineering

Supervised by

Associate Professor **Ren, Tongwei**

Software Institute

NANJING UNIVERSITY

Nanjing, China

April, 2018

摘 要

随着互联网技术的不断发展，利用教学支持系统开展大学教学活动成为了全球高校提升教学质量、提高办学竞争能力的必然选择和共同趋势。但目前大多数教学支持系统功能相对局限，仅能满足对一些日常教学活动的支持，如作业发布、作业上传、课件上传与下载等。而对于课程活动管理、课程资源管理、以及师生互动等一些比较复杂的功能，多数系统都还无法很好地满足。

在这样的背景之下，思目创意科技有限公司设计并开发了一款新型的基于“事务管理模型”的超级账号教学支持系统。该系统采用前后端分离的方案进行系统开发。系统前端主要由课程管理模块、课程活动模块、会话模块、小组模块、以及知识体系模块五个部分组成。其中课程管理模块由课程信息管理、课程人员管理、以及课程资料管理这三个子模块组成，小组模块由小组人员管理、小组活动管理、以及小组资料管理这三个子模块组成。

超级账号教学支持系统前端采用 MVC 模式进行架构设计，并由 WebSocket 以及基于微服务的服务端为前端提供数据支持。在技术实现上，该系统前端是基于 React 框架的单页面应用，具有页面加载速度快、组件功能强大、交互友好等特点。该系统前端以 React 为核心，并使用 Redux 框架、React-Router 框架、以及 Ant-Design 等技术进行组件化开发，其中 Redux 框架负责完成前端数据的管理和缓存工作；React-Router 框架负责前端页面路由的管理和匹配工作；Ant-Design 是一套丰富的组件库，系统前端在该组件库的基础上做了定制化开发。在项目构建方面，该系统前端通过 Babel 工具对代码进行编译优化，通过 Webpack 工具对项目进行依赖配置和模块打包；在部署方式方面，该系统前端使用 Docker 技术部署在公有云上。

使用上述方案开发的超级账号教学支持系统前端，具有很快的页面响应速度，很强的页面交互能力，面对操作较为复杂的使用场景，基于 React 的前端能够提供稳定的运行环境和很高的可用性。

关键词：单页面应用、前后端分离、React 框架、Redux 框架、Webpack

Abstract

With the rapid development of Internet technology, using teaching support system to improve teaching quality and educational competitiveness has become a common trend in university. However, functions of most teaching support system are relatively limited. They can only support some daily teaching activities, such as homework assignment, homework uploading, teaching materials uploading and downloading. For the more complex functions such as course activity management, course resource management, and teacher-student interaction, most systems are still unable to perfectly meet.

Against this background, SIMU Technology developed a new teaching support system designed by 'affair management model'. The system is based on seprate front and rear ends. The front-end is divided into five modules: course management module, course activity module, chat module, group module and knowledge module. The course management module consists of three submodules, which are course information management, course staff management and course references management. The group module also consists of three submodules, which are group staff management, group activities management and group references management.

The front end of SuperID Teaching Support System is designed by MVC pattern. Data is supported by WebSocket and the back end based on Micro-Service. The front end is a single-page-application based on React. It has a fast speed of page loading and is composed of a set of powerful components. With the help of React, the front end uses Redux to manage page states and uses React-Router to manage page routes. It also uses a lot of components provided by Ant-Design. In terms of deployment, the front end is built by Babel and Webpack and is deployed by Docker on a public cloud server.

With the above technology, the front end of SuperID Teaching Support System has a good speed in responding and a strong ability of page interaction. It is quite stable and can provide high level usability.

Keywords：Single-Page-Application, Front and rear separation, React, Redux, Webpack

目 录

图目录

表目录

第一章 引言

1.1 项目背景

随着互联网的发展，越来越多的学校建立起了自己的教学支持系统，用于辅助师生的日常教学活动。教学支持系统不仅使得教学方式不再受时间和地域的限制，也使得教学资源的统一管理得到优化，为师生提供了虚拟化和集成化的教学环境。教学支持系统能够让老师教得更轻松，让学生学得更方便，也让教学管理者能够更合理更优化的分配资源和管理资源，并最终提升办学质量，提高学校竞争水平。目前大多数的教学支持系统，其实现基本都是基于 J2EE 或 LAMP 等一些比较传统的技术方案。随着网络教育的飞速发展，这些系统将越来越不能适应日渐复杂的教学需求。因此，建立新型的的教学支持系统显得越来越重要，也正在成为教育技术领域许多技术专家所专注的课题。

新型的教学支持系统应当具有以下几个特点：能够灵活地对教学资源进行管理和分享、能够有效地记录并分析课程活动的开展过程、能够加强师生围绕课程主题的互动性、以及能够提高老师和助教的作业批改与统计的效率等等。另外，在目前的教学场景中，许多课程都会涉及到小组活动，但对小组的组织和管理相对比较松散，没有一个统一的工具进行管理支撑。因此，新型的教学支持系统还应当对课程小组进行集中管理，管理的内容包括小组人员、小组内部活动、以及小组内部资料等等。除此之外，目前高校与企业之间的合作也越来越频繁，新型的教学支持系统还应当为校企合作提供平台支撑，打破高校与企业之间的边界限制，并最终推动教育社会化的发展。

本文所设计的教学支持系统，是思目创意科技有限公司（以下简称思目）旗下的一款产品。该产品使用思目的“事务管理模型”进行设计，这套模型以“事务”作为最小管理单位，将所有与该事务相关的“人”、“财”、“物”组织到一起进行统筹调度。在“事务管理模型”的驱动下，超级账号教学支持系统打破了传统教学支持系统的管理模式。系统内每一门课程就是一个“事务”，参与到课程中的老师、学生、助教就是事务中的“人员”，课程所涉及的教学经费、教学资源就是事务中的“财”和“物”。从长远的角度来看，该系统不仅可以提升教学

质量，还能为以后的校企合作、学校与学校之间的合作提供平台支撑，使得合作更方便、方式更透明、沟通更高效。

1.2 国内外发展现状

1.2.1 国外发展现状

教学支持系统的概念最先是在信息化发展速度较快的欧美国家中诞生。比如由哥伦比亚大学计算机科学系开发的 WebCT 系统、印第安纳大学的 Oncourse 系统、麻省理工学院的 Stellar 系统以及斯坦福大学的 CourseWork 系统等等[吴伟敏, 2002]。后来随着开源软件的发展，出现了 Sakai 和 Moodle 这两款比较著名的教学支持系统。

Sakai 用 Java 编写，于 2005 年发布第一个版本，目前已有全球超过 300 所高校使用该系统。它支持插件结构，可以由用户自由定制服务范围，这其中包括了课程管理系统中很多常见的功能，如文档发布，成绩册，讨论区，聊天室，作业上传，在线考试等等[Berg, 2009]。但 Sakai 强调自主、协作的学习模式，它的设计理念更适用于国外高校，所以目前国内仅有南方科技大学、重庆大学等院校使用该系统。

Moodle 基于 PHP 和 MySQL 数据库进行设计，由澳大利亚人 Martin Dougiamas 主持开发。据统计，目前已经有超过 45000 个已注册网站，为 3200 万位用户提供约 300 万个课程服务。Moodle 对课程具有强大的管理功能，能够灵活地对课程进行权限设置以及状态设置，并且能够有效的追踪学习进度，包括作业提交截止日期等[Building, 2005]。与此同时，Moodle 也存在着页面结构不合理，缺乏导航信息，资源不能共享等一些明显劣势。

1.2.2 国内发展现状

相较于欧美国家，国内教学支持系统的发展还处于初期探索阶段，虽然发展趋势缓慢，但仍保持着上升的态势。许多高校所使用的内部系统大多都借鉴了上述几个国外案例的框架与思路。如复旦大学的 eLearning 系统，在 Sakai 系统的基础上添加了大量的本地化开发，产生了 Sakai 复旦大学共享版。同时，结合我

国自身的教育模式来看，盲目地使用国外系统也存在着许多与国情不符的地方，包括了使用习惯、学习方式、课程组织结构等无法兼容的问题。

目前国内自主开发的比较成熟的有清华大学网络教育系统、北京师范大学网络教育系统、以及南京大学软件学院教学支持系统等，他们各有特色也都存在着明显的不足之处。

清华大学的系统包括了网络资源、课程课件资源以及资料库等组件，是一个先进的网络教育基础服务支撑系统。该系统功能较为简单，仅能够满足教学资源管理的需求，更像是一个文件存储服务器。系统内缺少课程管理的概念，因此也就无法满足对于课程教学活动的支持。

北京师范大学的系统由课程资源、课件展示、精选课程、资料库等模块组成，该系统强调系统方法论和学习理论的构建，便于系统用户建构自己的知识体系架构。该系统在满足了教学资源管理的基础之上，加入了教学资源与用户知识体系相结合的功能，是清华大学的系统的改进版本。但该系统也同样缺少了对于课程信息以及课程活动的管理功能，老师和学生在系统中的角色区分不够明显，且每个用户的知识体系仅对自己可见，无法与系统其它用户进行分享。另外，该系统在交互能力和性能方面也相对比较薄弱。

南京大学软件学院现有的教学支持系统由 TSS 和 CMS 两个版本组成。TSS 包含了基本的课程模块、作业模块和留言板模块，其功能相对单一，系统用户能够通过 TSS 系统查看课程信息，提交课程作业，并通过留言板参与课程讨论。另外，TSS 系统所采用的技术相对较为传统，界面也比较粗糙，无法适应日渐复杂的教学活动需求。CMS 版本是在开源软件 Moodle 的基础上做的定制化开发，其基本功能与 Moodle 类似。与 TSS 系统相比，CMS 系统功能更丰富，技术更先进，但也同样存在着页面结构不合理、导航信息不够清晰等缺陷。另外，虽然 Moodle 为 CMS 系统提供了强大的定制化能力，但其大多数设计理念还是基于国外的教学场景，在课程组织方式和使用习惯上，CMS 系统还无法与国内的教学场景完全契合。

这些国内现存的教学支持系统，普遍存在着功能泛化，功能模块冗余等缺陷。这些系统在造成资源重复建设的同时，没有充分发挥互联网平台所能提供的协作能力和自主能力。而其他的一些产品大多功能较弱，如其中的大部分缺少管理和

评价系统，以及课程开发系统等，还有的一部分仅仅支持同步教学方式，使用场景较为局限[吴伟敏, 2002]。

上述所提到的这些国内外教学支持系统，通过版本的迭代都在一步步的更新与完善，这其中汇聚了大量教育技术专家的心血和努力，这些有益探索所带来的经验和教训值得学习借鉴。

1.3 本文主要工作

本文设计并实现的超级账号教学支持系统前端，以超级账号教学支持系统为背景。为了能够满足日益丰富的教学活动需求，提供更强大的页面交互能力，该系统采用前后端分离的方式进行开发，前端页面内容丰富，功能强大。通过该系统前端，用户能够对课程信息、课程资源、课程活动、课程小组、以及个人知识体系进行统一管理。另外，前端还提供了强大的会话功能，能够有效地提升师生的互动能力。

超级账号教学支持系统前端是基于 React 框架的单页面应用。该系统前端采用组件化的思想进行开发，前端内的所有页面都是由可复用的组件组成。另外，前端采用 Redux 框架进行数据管理以及数据缓存，通过 React-Router 框架完成页面路由管理与路由匹配，使用 Babel 进行代码编译和语法转换，最终通过 Webpack 进行工程化打包构建，并使用 Docker 进行项目部署。使用上述技术设计并实现的超级账号教学支持系统前端，具有交互友好、性能优秀的特点，并且能够做到在浏览器端进行页面按需加载和静态资源缓存，在一些环境苛刻的使用场景中，系统前端也能够保证最起码的可用能力。

1.4 论文组织结构

本文的组织结构如下：

第一章 引言。本章介绍了超级账号教学支持系统前端的项目背景、国内外在教学支持系统方向的发展现状、本文的主要工作、以及论文组织结构。

第二章 相关技术综述。本章介绍了项目所使用的核心技术和框架，包括单页面应用、React 体系相关技术、SASS、ECMAScript 6、Fetch、前端工程化相关技术、以及 Express 框架。

第三章 超级账号教学支持系统前端的需求分析。本章提出了项目的整体需求，并对系统前端的功能性需求和非功能性需求进行了分析。

第四章 超级账号教学支持系统前端的设计。本章对系统前端的总体设计思路进行了概述，并将前端划分成五个大块，依次阐述每一块的设计思路。

第五章 超级账号教学支持系统前端的实现。在详细设计的基础上，本章重点阐述了课程资料模块、课程活动模块、会话模块、以及富文本编辑组件的具体实现细节。

第六章 总结与展望。本章总结了论文期间所做的工作，并且就超级账号教学支持系统前端的未来扩展作了进一步展望。

第二章 相关技术综述

2.1 单页面应用

单页面应用（Single-Page Application）是指只有一张 Web 页面的应用，是目前业内较为流行的网络应用程序构建模型。与传统网站打开多个页面的方式不同，单页面应用使用动态重写当前页面的方式来与用户进行交互，是一种从 Web Server 加载的富客户端[Flanagan, 2006]。

单页面应用进行路由跳转时，页面上仅有局部的一些资源需要被重新刷新，而像一些公共的页面资源，只需在网站第一次呈现时被加载即可。这大大提升了 Web 应用的交互体验，避免频繁打断用户，使得网站的表现形式更像是一个桌面型应用。与多页面应用相比，单页面应用开发成本较高，需要借助专门的框架进行搭建，更注重模块化的开发和设计。但维护成本相对更低，页面转场更流畅，能够带来更优异的用户体验。

超级账号教学支持系统前端页面内容丰富，功能强大，从业务和技术实现的角度来看，使用传统的后端页面直出方案或是多页面应用的形式都是不合适的。因此，单页面应用是前端呈现方式上的最优选择。

2.2 React 体系相关技术

从狭义的角度来说，React 只是一套 JavaScript 框架，但从广义的角度来看，React 并不仅仅是这套框架本身，它代表了一个完整的前端技术生态体系。React 体系不仅在 Web 端、移动端、服务器端有所应用，甚至在目前火热的 VR 领域也都有涉及。所以 React 所代表的已不再是一个单纯的前端框架，随着背后生态体系的逐渐发展，它已成长为行业内关于 Web 应用的通用解决方案。超级账号教学支持系统基于 React 进行开发，本小节将逐一阐述系统所使用到的各项 React 相关技术。

2.2.1 React

React 是一套开源的 JavaScript 框架。它定义了 HTML 页面在浏览器端的渲染机制，当页面数据发生改变时，React 可以高效地对页面进行重新渲染。它起源于 Facebook 的内部项目，由软件工程师 Jordan Walke 创建，首先应用在 Facebook 的 newsfeed。之后在 2012 年，基于 React 的著名社交产品 Instagram 问世，并于次年在 JSConfUS 开源[陈屹, 2016]。

React 框架具有声明式和组件化两个显著特点，开发者以声明式编写 UI，使得代码更可靠，更利于调试[Bertoli, 2017]。对于组件化，React 以组件的方式构建每个页面，每个组件独立维护自己的内部状态。通过组件化，前端项目无需再使用模板代码，能够更合理地管理页面数据以及数据在页面之间的传递。严格来说，在经典 MVC 架构中，React 作为前端框架只是完成 View 视图层的工作，之所以这么做，是因为 Facebook 认为，面对大型前端项目，如果把 MVC 所有的工作都放在一起，代码量将变得极为庞大，难以维护和扩展。所以 React 框架本身只需专注页面渲染，由 React 体系内的其他框架来完成 MVC 模式中的其他工作。

与其他前端框架不同的是，React 在页面渲染上使用了“虚拟 Dom”技术[严新巧, 2017]。传统的 Web 应用中，如果页面内容发生了变化，会立即触发浏览器对整个 HTML 页面的重绘动作。重绘过程中所有的 Dom 元素都会被重新创建，创建完成之后浏览器再对所有元素进行重新渲染。而在单页面 Web 应用中，通常只会对页面上的部分内容进行更新，也就意味着只有部分 Dom 元素需要被重新渲染。所以如果频繁地对整个 Document 进行重绘会造成严重的性能损失。React 的“虚拟 Dom”技术是指，React 会在浏览器内存中将整个 HTML Document 结构以 JavaScript Object 的形式存储下来，形成一个虚拟的 Document。当页面内容发生变化时，React 并不会立即把整个 HTML Document 都重绘一遍，而是先把内容变更部分更新到虚拟的 Document 中，然后再通过 Diff 算法比较出虚拟 Document 改变前与改变后的的差异，最终将差异的部分渲染到真实的 HTML Document 中。这样做的好处是：当页面只有部分数据发生变化时，React 通过计算虚拟 Dom 的差异，实际上只会对部分 HTML 节点进行重新渲染，其余大部分不变的内容无需进行重绘操作。这在单页面应用中，能够有

效地提升页面渲染性能。

2.2.2 Redux

随着单页面应用的规模越来越大，开发日趋复杂，前端页面所涉及的数据状态管理越来越多。这些数据可能是向服务器请求来的数据，也可能是页面在浏览器中的缓存数据，或是本地生成的还未持久化到数据库的数据，还可能是页面变化所产生的大量 UI 状态数据。由于 React 只是一个 View 层框架，所以对于数据状态的管理，Facebook 提出了 Flux 架构，Redux 正是基于 Flux 架构所设计的开源状态管理框架。Redux 受函数式编程语言 Elm 的启发，由 Dan Abramov 和 Andrew Clark 于 2015 年共同创建[邓世超, 2017]。该框架在 Flux 的基础上进行了优化和改造，使得状态管理逻辑更简单清晰。

Redux 遵循三大基本原则：单一数据源、State 是只读的、使用纯函数来执行 state 的变更[Daniel, 2017]。具体来讲，整个应用的 state 将被存储在一棵状态树中，应用中的每个组件都可访问这个数据源，得益于树形结构的存储方式，数据源也是可以进行模块化设计的。Action 函数是变更状态树的唯一方式，这使得所有的修改都能够被集中化处理。在超级账号教学支持系统中，Redux 不仅仅是一个状态管理容器，它还对来自服务器的数据进行缓存处理，这使得用户无需频繁访问服务器进行数据请求，提升了页面渲染数据的速度。

2.2.3 React-Router

React-Router 是 React 体系中的路由解决方案，也是 Facebook 官方提出的唯一可选方案。该框架通过简单的 API 保持页面 UI 与 URL 的同步，它包含了页面按需加载、动态路由匹配、以及建立正确的位置过渡处理等一系列强大的功能。页面的按需加载功能能够提升大型网站的页面加载速度。在前端页面的渲染过程中，React-Router 只会将用户当前所关心的页面代码加载到浏览器中，用户改变页面 URL 之后，React-Router 会根据开发者定义的路由匹配规则，动态地将页面后续需要渲染的页面代码加载到浏览器中。

React-Router 使用路由嵌套的概念来定义页面的路由集合，并使用树形结构进行路由存储。用户在浏览器中进行页面切换时，React-Router 会按照深度

优先算法遍历整个路由集合，来确定路由该如何进行跳转。对于两个或多个同一层级的路由而言，React-Router 会按照路由定义的顺序自顶向下地进行路由匹配。另外，路由在配置的过程中，React-Router 提供正则表达式语法，使得配置更灵活。

2.2.4 Ant-Design

Ant-Design 是由阿里巴巴蚂蚁金服推出的 UI 组件库，也是目前 React 体系中较为成熟和流行的 UI 解决方案。它更像是一个服务于企业级产品的设计体系。它基于“确定”和“自然”的设计理念，通过模块化的解决方案，让设计者更加专注于用户体验的提升。Ant-Design 本身由 React 框架编写，封装了大量在单页面应用中会使用到的交互组件，包括了上传组件、导航菜单、对话框、表单组件、消息通知组件等等。另外，Ant-Design 提供便捷的主题定制方案，对于复杂项目来说大大降低了开发成本。

超级账号教学支持系统前端在 Ant-Design 提供的组件基础之上，对主题风格做了定制化修改，给用户呈现了精致流畅的页面操作体验。

2.2.5 Draft.js

Draft.js 是 Facebook 官方推出的富文本编辑器解决方案。本质上，它并不是一个单纯的富文本编辑器，其灵活的 API 使得开发者能够根据业务需求自定义编辑器功能。超级账号教学支持系统在课程活动模块中提供活动创建功能，用户需要在前端页面上进行富文本操作，且需要在编辑器区域内发送物资和资金这些非传统的编辑内容。业内现有的富文本编辑器产品都无法满足灵活的自定义需求，且这些同类产品在数据存储方面，大多都采用了 HTML 字符串直接写入数据库的方式。这种方式会造成大量的存储空间浪费，也无法灵活地对文本内容进行提取和筛选。Draft.js 将文本内容与内容格式分开，使用结构化的数据替代纯 HTML 字符串进行存储。这能有效减少内容冗余，节省数据库存储空间，也提升了页面向 Server 请求数据的速度。另外，课程活动模块中提供活动详情的版本对比功能，使用结构化数据能够方便地将活动内容提取出来进行差异比对，这在传统的富文本编辑器中也是无法做到的。

2.2.6 Immutable.js

Immutable.js 是一个完全独立的 JavaScript 库，它弥补了 JavaScript 数据结构都可变的问题。在 React 应用中，页面常常涉及频繁的状态修改，无论是组件自身的状态还是 Redux 数据源中的状态，使用 Immutable.js 能够使得这些变更更快、更安全。它的内部实现了一套完整的持久化数据结构，包括了 List、Map、Set 等，API 也与原生的 Object 或 Array 对象类似。其核心思想是，数据一旦被创建，就不能被更改，每次对该数据的变更都会产生一个新的数据对象。与此同时，为了避免数据深拷贝所带来的性能损失，Immutable.js 采用结构共享的算法进行设计。如果对象树中的一个节点发生变化，只修改该节点和受它影响的父节点，其他节点则进行共享。

在超级账号教学支持系统项目中，Immutable.js 与 Redux 搭配使用，能够有效地提升前端页面处理数据的速度，也避免了很多 JavaScript 原生数据结构所带来的风险和隐患。

2.3 Sass

Sass 是一种 CSS 扩展语言。传统的 Web 项目中，编写 CSS 样式代码通常是一件较为繁琐的事情。且与其他编程语言不同，CSS 代码上下文之间通常没有很强的逻辑关联。所以当代码规模逐渐扩大时，对 CSS 代码的管理成本也逐步增加。Sass 提供了一种接近于高级编程语言的方式来编写 CSS 样式代码。它能够像其他语言一样，将 CSS 样式声明成普通变量进行引用，也能够引入其他 Sass 文件中公共变量。Sass 在兼容 CSS 所有特性的前提之下，提供很多 CSS 没有的特性来书写样式。它允许在样式语句中使用算式对样式进行计算，且算式中可以使用变量。Sass 也允许 CSS 选择器的嵌套写法，使得 CSS 选择器的先后次序更清晰。Sass 同时还提供了继承机制，能够允许一个选择器继承另一个选择器，减少了重复代码的书写。除此之外，Sass 还提供了颜色函数、条件语句、循环语句、自定义函数等一些高级编程语言才会有的语言特性。这些特性使得样式编写更便利，也使得样式代码结构更清晰、更易于维护。

Sass 还提供专门的解释器 Sass-Loader 对 Sass 脚本进行解析编译。所有

的 Sass 脚本在项目打包构建的过程中，都会被 Sass-Loader 编译成浏览器可以理解的 CSS 代码。另外，开发人员在开发过程中，Sass-Loader 会对脚本进行即时监控，无需刷新页面即可更新页面样式，从而实现样式更新的热加载。

超级账号教学支持系统前端页面数量众多，涉及到丰富的样式代码。使用 Sass 扩展语言进行开发能够提升开发效率，提高 CSS 样式代码的可读性，降低代码维护成本。

2.4 ECMAScript 6

ECMAScript 6（简称 ES6）是 JavaScript 语言的新一代标准，正式发布与 2015 年。它的出现使得 JavaScript 语言成为企业级开发语言，赋予了 JavaScript 编写复杂大型 Web 应用程序的能力[阮一峰, 2017]。1997 年，ECMA 标准化组织发布了 JavaScript 语言的第一版标准文件，规定了浏览器脚本语言的一系列标准，至今已发展到第六版。ES6 版本中，对 JavaScript 原有的一些语法进行了扩展的同时，引入了大量的新型语法功能，这里面包括了 let 和 const 命令、Proxy 和 Reflect 机制、Promise 对象、Class 机制、Module 机制等等。这些新特性使得 JavaScript 不再是一门简单的脚本语言，也使得越来越多的大型互联网应用应运而生。超级账号教学支持系统前端完全使用 ES6 语法编写，本小节将重点阐述系统所使用到的两个 ES6 机制。

2.4.1 Promise 异步调用

Promise 是异步编程的解决方案。传统 JavaScript 在处理异步调用时，采用回调函数和事件监听的方式，这在大型项目中容易造成“回调地狱”等问题，使得代码结构混乱，难以维护。相比之下，Promise 的出现使得异步编程结构更清晰更合理，它最早由开源社区提出和实现，后来 ES6 将其写进了语法标准中，在 JavaScript 中提供了原生的 Promise 对象[阮一峰, 2017]。

Promise 对象提供了一个容器，里面保存了将来某个时刻结束的事件结果，这就是异步调用的基本逻辑。该对象的状态不受对象外的状态影响，一个 Promise 就代表了一个异步操作。每个 Promise 都有进行中、调用成功、调用失败这三种状态，然后由异步事件的操作结果决定了当前的 Promise 处于哪个状

态中，这与操作系统中状态机的概念类似。另外，一旦 Promise 的状态发生改变，那么 Promise 将进入固定状态。这意味着，程序可以继续进行当前与 Promise 无关的一些操作，无需中断当前行为来响应 Promise，未来的任何一个时刻都可以从 Promise 对象中取回结果。这与 JavaScript 的事件监听机制有着显著区别，事件监听要求一旦 Event 被触发，监听器就需要立即进行处理，程序当前的操作也因此被中断，如果事后再去处理监听器，是得不到事件结果的。超级账号教学支持系统前端所有的异步请求调用，都使用了 Promise 机制。

2.4.2 Module 机制

在 ES6 标准出现之前，JavaScript 是没有模块体系的，这所带来的一个明显问题就是，前端项目无法将一个大文件拆分成几个相互依赖的小文件，然后再通过一定的方式组合起来，这对于开发大型前端项目来说，增加了项目的开发难度和维护成本[Nicholas, 2017]。开源社区在 ES6 之前也制定了一些模块加载方案，包括了用于服务器端的 CommonJS 以及用于浏览器端的 AMD 这两种。相比之下，Module 语法实现原理更简单，成为了服务端和浏览器端的通用解决方案。在设计思想方面，Module 要求尽量的静态化，即在程序的编译阶段完成模块相互依赖关系的确定[Silva, 2015]。CommonJS 和 AMD 这两种方案，都只能在程序运行阶段才能确定模块依赖关系。另外，Module 机制使得程序内一些函数的输入和输出在编译阶段就能预先确定，这也是上述两种方案所做不到的。

使用 ES6 的 Module 机制，能够充分使用模块化的思想来开发超级账号教学支持系统，这提升了系统的可扩展性，降低了开发难度和维护成本。

2.5 Fetch

在传统的前端项目中，HTTP 请求大多由 ajax（XMLHttpRequest）完成，它的 API 设计比较粗糙，不符合关注分离的原则，配置和调用方式也比较混乱。为了解决 ajax 所带来的问题，JavaScript 提供了新的替代方案——Fetch。Fetch 是基于 Promise 设计的，在 IE8 及更高版本的现代浏览器中都能稳定运行。首先在写法上，Fetch 的 Promise 语法大大优于 XHR 基于事件的异步模型。Fetch 语法简介，更加语义化，对于开发者来说能够更加便利地处理 HTTP 异步请求。

另外，当接收到一个代表错误的 HTTP 状态码时，从 Fetch 返回的 Promise 不会被标记为 reject 状态，当且仅当网络连接中断或请求被阻止时，才会被标记为 reject，这使得请求的逻辑更合理，前端对于请求失败的处理更可控。且默认设置下，Fetch 不会从服务器端接受或向服务器端发送任何 Cookies，避免了 XHR 所带来的 Cookies 跨域问题。另外，Fetch 对跨域资源访问和 HTTP 扩展方面也有良好的支持。

2.6 前端工程化相关技术

前端工程化是前端架构中很重要的一环，随着目前 Web 应用的规模日趋庞大，前端项目已经不再是过去 HTML、CSS、JQuery 插件的简单组合了[周兴宇, 2016]。本质上来说，现在的前端工程与其他的软件工程是属于同等概念的[周俊鹏, 2018]。在超级账号教学支持系统中，使用 React 以及 ES6 语法进行页面的组件化开发，使用 Sass 语言编写页面样式，这些都需要特定的解释器来对项目文件进行工程化的解析和变异。本小节将从模块化、规范化、以及自动化这三个方面来阐述超级账号教学支持系统在进行前端工程化所使用到的相关技术。

2.6.1 NPM

NPM（Node Package Manager）是 Node.js 默认的包管理系统。它完全用 JavaScript 编写，最初由 Isaac Z.Schlueter 进行开发设计，初衷是为了解决 JavaScript 以及前端项目中所遇到的模块管理困难、模块依赖混乱等一些问题[Aquino, 2016]。NPM 会基于前端项目的 package.json 文件，来管理项目所需模块，并自动维护依赖情况。package.json 文件中可以指定每个模块的任意可用版本，这样避免了模块更新所带来的兼容性问题，也使得模块能够向最高可兼容版本进行自动更新。并且 NPM 官方维护了一个开放的公共仓库，任何用户都可以按照“先到先得”的原则注册自己的模块，这使得前端项目的模块化管理更轻松，结构更清晰合理，实现了前端工程化中的自动化需求。

2.6.2 Babel

超级账号教学支持系统前端完全采用 ES6 语法进行编写，这对于一些还没有完全支持 ES6 所有语法特性的浏览器来说，可能会带来兼容性的问题，而 Babel 就是一个解决该问题的 JavaScript 编译器。它能够将项目代码统一编译到一个更通用的语言规范版本（例如 ES5）。以 ES6 中的箭头函数为例，Babel 会把箭头函数转换为浏览器能够理解的普通函数写法。由于 Babel 只是完成语法转换的相关工作，所以对于 Promise 或是一些比较新的 JavaScript 原生方法，Babel 也提供了相应的 Babel-Polyfill 来进行语法支持。Babel 使得开发者在使用最新语法特性的同时，无需担心浏览器的兼容性问题，从而实现了前端工程化中的规范化需求。

2.6.3 Webpack

Webpack 是目前较为流行的前端项目构建部署工具。本质上，它是一个 Web 应用静态模块打包器[吴浩麟, 2018]。在打包的过程中，Webpack 会递归地构建项目的依赖关系图，梳理出项目所需要的每个模块，然后把这些模块进行打包，输出一个或多个打包后的 bundle 文件[Tuomas, 2017]。在确定好模块的依赖关系之后，Webpack 会在特定解释器的帮助下，对一些扩展语言进行解析编译。比如 Sass-Loader 对 Sass 的解析过程，就是在 Webpack 打包构建的时候完成的；项目中用到的静态资源文件也会通过 File-Loader 解析成 JavaScript Blob 对象插入到代码中。另外，Webpack 在打包的过程中，还会对项目代码进行优化，例如代码压缩、代码丑化、代码分割等一系列操作。代码压缩和代码丑化能够有效地减少项目体积，提升页面的首次加载速度。代码分割会将不同模块的代码进行独立打包，实现模块的按需加载，满足前端工程化中的模块化需求。

2.7 Express

Express 是一个轻量级的 Node.js Web 框架。它基于 Node.js 系统，提供快速、开放、简单的方式来帮助开发者构建 Web 应用[占东明, 2016]。Express 框架本身体积轻便，只是在 Node.js 系统的基础上加入了 Web 应用所必须的一些

功能，并且保留了 Node.js 的完整特性。Express 提供丰富的 HTTP 工具和中间件，开发者可以灵活地使用各种中间件来快速构建 Web 应用。

超级账号教学支持系统前端作为前后端分离的 Web 应用，在打包构建完之后，最终会运行在 Express 提供的 Web 服务器上。Express 会监听指定端口下的所有浏览器请求，然后根据请求路径，将不同路径下的资源返回给浏览器。Express 由于足够轻便灵活，所以可以很方便地与 Nginx 结合起来，在项目进行线上部署的时候完成反向代理等操作。这也是超级账号教学支持系统前端选择 Express 作为 Web 服务器的一个很重要的原因。

2.8 本章小结

本章节综述了超级账号教学支持系统前端所用到的核心技术，并结合系统前端的具体使用场景，阐述了每项技术的使用理由。本章首先介绍了单页面应用的定义与优势，接着介绍了 React 体系相关技术、Sass 技术、ECMAScript 6 的核心技术、用于进行数据请求的 Fetch 技术、前端工程化的相关技术、以及支撑项目运行的 Express 框架。

第三章 超级账号教学支持系统前端的需求分析

3.1 超级账号教学支持系统整体概述

超级账号教学支持系统沿用了思目的“事务管理模型”。该“事务管理模型”是一种通用的管理模型，思目希望通过该模型解决以下三个问题：组织机构的管理模式僵化、组织及组织内部的边界制约组织发展、组织目标执行过程不明确。基于以上三个问题，该模型以事务为中心进行无边界管理，从而最大化资源价值，更有利于组织目标的实现，推动组织的发展。

具体而言，该系统以课程为管理中心，每一门课程对课程信息、课程内的人员、以及课程资料进行集中管理。针对每一门课程来说，在现有的教学场景中，课程目标的实现过程往往没有一个完整的记录过程，师生大多更看重最后的考试结果，而完整的活动过程记录有利于师生进行经验分享。所以，系统内提供基于主题的活动发布体系、基于角色的人员体系、以及基于课程的会话体系等一系列模块对课程活动进行管理支撑。在此基础之上，系统还提供了小组体系对课程小组进行管理，以及知识体系来帮助用户构建和分享个人的知识体系框架。

另外，目前的高校内部各部门与各院系之间、高校及高校之间、以及高校与企业之间，都存在着明显的边界限制，教学资源的流通与共享大多较为闭塞。尤其在高校与企业之间，一方面高校需要企业丰富的实践经验，另一方面企业也很迫切需要高校的人才资源，而目前市面上缺少相应的平台为这两者创造良好的合作环境。超级账号教学支持系统的无边界“事务管理模型”，能够打破传统的教学系统管理模式，为校企合作提供平台支撑，从而最大化课程价值，提高教学质量，推动教育社会化的发展。

超级账号教学支持系统的用户可以是老师、学生、学院教务员、学校管理者等，也可以是希望与学校进行合作的企业内部人员，或者是学术界的讲师、教授等等。使用该系统的用户可能是对教学管理系统不熟悉的用户，所以系统前端界面提供友好的导航以及操作提示。

3.2 超级账号教学支持系统前端的功能性需求分析

根据项目负责人和软件学院部分老师的共同讨论，超级账号教学支持系统的系统用户主要分为四大类：教务员、老师、学生和助教。根据相关负责人的描述，画出以下系统用例图。如图 3.1 所示，超级账号教学支持系统通过事务管理模型，主要实现了以下几个方面的功能：课程管理功能、课程活动功能、小组管理功能、课程会话功能、以及知识体系构建功能。其中课程管理包括课程信息管理、课程资料管理以及课程人员管理。

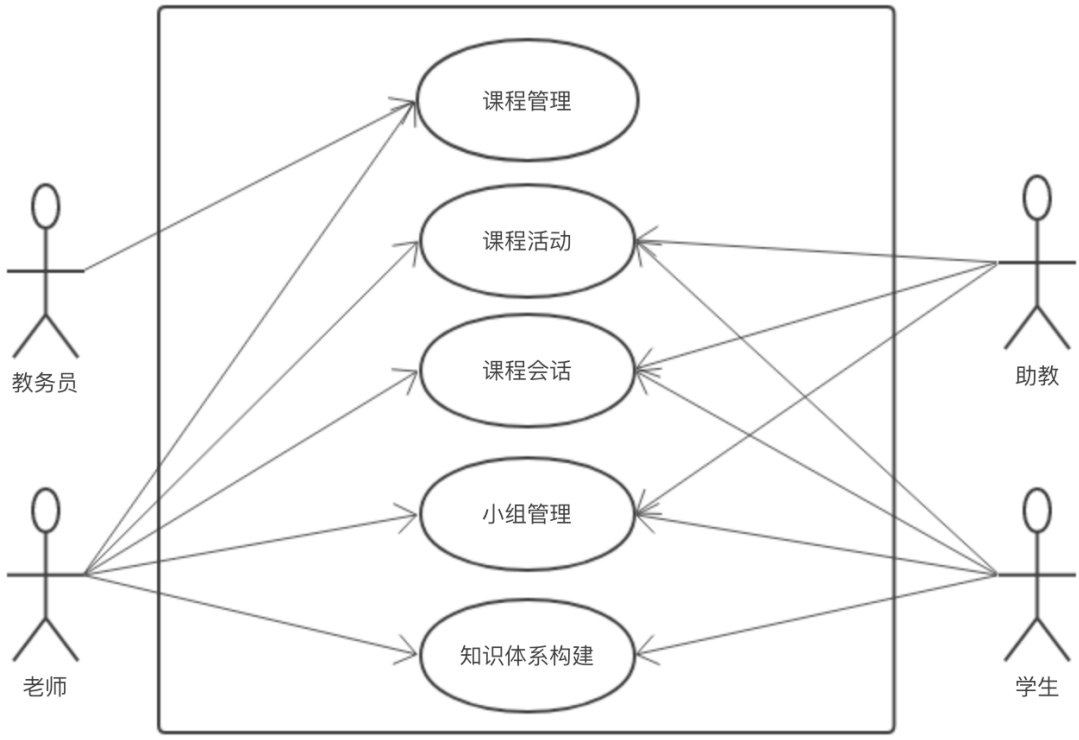

图 3.1 超级账号教学支持系统前端的总体用例图

在课程信息管理方面，教务员通过该系统可以在自身院系内进行课程创建。每一门课程都有自己的首页，在首页上可以清楚地了解这门课的简介、授课内容等基本信息。学生可以根据学院要求和自身兴趣在课程首页中进行选课。选课同时提供选课码自主选课和任课老师批准申请这两种方式。

在课程资料方面，该系统支持多级目录的文件库功能。所有课程内相关的课件、教材、以及作业素材等，都会存储在文件库中。老师和助教可以在文件库中进行资料上传，学生可以直接在页面上预览或是下载到本地，也可以对文件进行分享操作。该功能使得课程资源的分享更自由，教学资源存储更集中。

在课程人员管理方面，在事务管理模型中，每个事务都拥有一个角色体系。对应地，每门课程主要涉及到教师、学生和助教这三大角色，该模块可以对这三类角色进行人员分配。每个角色都拥有一个角色卡片，上面记录了用户的基本信息。用户也可以通过角色卡片查看该角色在这门课程内的历史活动记录。通过这种方式，课程参与者就能很清楚地了解到每个角色在课程内的活动轨迹，这使得周期性的总结和奖项评定工作能够有据可依。

在课程活动方面，每一门课程都有自己的“课程活动”专栏，任课老师可以在这一栏内发布基于某一特定主题的课程活动。以使用频率最高的作业为例。老师发布作业之后，学生可以在活动详情中查看作业要求，然后在作业规定的时间内进行作业提交。助教则根据学生的提交结果进行作业批改，并最终在系统内显示相关作业的统计信息，如最高分、最低分、以及平均分等统计数据。教师也就可以通过这些数据来分析课程活动的完成情况以及对教学质量进行评估。另外，每个活动内都提供评论功能，课程内的所有角色都可以在该区域内围绕该活动主题进行沟通讨论，所有的活动过程也都有了相应的记录。

在小组活动方面，学生在课程中可以自由创建小组，小组在模型上也是一个事务的概念，每个小组也都拥有自己独立的发布体系，人员角色体系以及文件库体系。小组的出现让涉及到小组活动的课程能够更好地跟踪每个小组的任务完成情况，也使得小组内的互动、活动记录以及资源整理更有条理。

在课程会话方面，每个课程内部都有基于本课程主题的会话模块。课程内的所有角色都可以在其中进行与课程主题相关的会话。该模块包括私聊和讨论组群聊两种会话形式。另外，系统会保存所有用户的会话记录，用户可以根据关键字或是内容类型进行会话记录检索。

在知识体系构建方面，超级账号教学支持系统会对每门课建立知识索引，进行知识分类，给学生提供了一个更宏观的视图来审视自己的知识获取情况。因此，学生可以根据自身发展的需求来针对性地学习课程，从而使自己的知识体系更加完整。每个知识体系都是一个树状结构，用户可以在知识体系中添加标签，并把标签与相关的课程关联起来。这样用户就可以清楚地看到自己的知识获取轨迹。另外，完整的知识体系还能够起到经验传承的作用。系统为每个学生各自的知识体系提供了收藏、评论、以及分享等功能。

3.2.1 课程管理前端的功能性需求分析

图 3.2 是课程管理功能的用例图。在课程管理模块中，主要分为课程信息管理子模块、课程人员管理子模块、以及课程资料管理子模块三个部分。教务员和老师是课程管理模块的主要参与者。

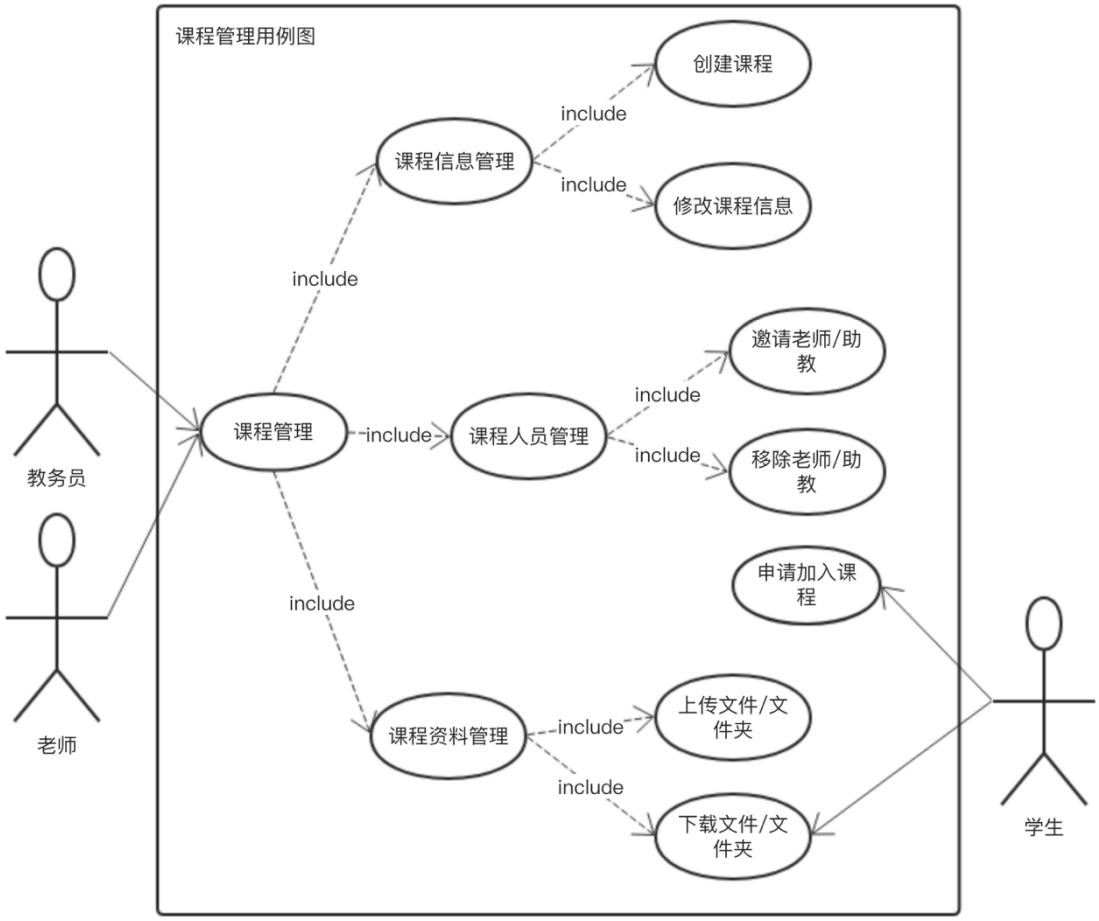

图 3.2 课程管理用例图

在课程信息管理子模块中，教务员可以创建课程，课程创建完毕之后，教务员和老师可以对课程信息进行修改。创建课程和修改课程信息在功能逻辑上基本相似，以创建课程为例，给出该需求的用例描述，如表 3.1 所示。

表 3.1 创建课程的用例描述

ID	1
名称	创建课程
参与者	教务员
描述	每个院系的教务员在学期开始前需要创建本院系课程
优先级	高
触发条件	用户在所有课程列表的下方点击创建课程按钮

前置条件	用户已登录，并鉴定为教务员身份
后置条件	无
正常流程	1. 用户点击创建课程按钮 2. 用户填写课程基本信息，包括课程名、课程编号、学分等 3. 用户指定一位或多位任课老师 4. 用户提交表单 5. 页面重新渲染课程列表，在列表中添加新创建的课程
分支流程	1. 在用户指定任课老师时，可以进行关键字搜索 2. 输入关键字，可以是老师姓名或者是老师工号 3. 显示搜索结果并选择
异常流程	1. 输入系统中已存在的课程名 2. 系统提示课程名已存在
其他	表单中有字段忘记填写时，阻止用户提交，进行错误提示

在课程人员管理方面，课程创建完之后，教务员和现有任课老师可以邀请新的任课老师加入课程，任课老师还可以邀请非本课程学生的人员来担任助教。对于必修课，系统将自动加入相应学生到课程中；对于选修课，有两种加入方式：通过选课码自主加入，或者发送选课申请等待老师批准后加入。学生加入课程的具体方式如表 3.2 所示。

表 3.2 学生加入课程的用例描述

ID	2
名称	学生加入课程
参与者	学生、老师
描述	学生可以浏览所有课程列表，查看课程基本信息之后，选择加入课程
优先级	高
触发条件	用户在课程首页点击申请加入课程按钮
前置条件	用户已登录，并鉴定为学生身份，课程类型为选修课
后置条件	无
正常流程	1. 学生浏览课程基本信息，点击申请加入课程按钮 2. 学生填写申请理由 3. 老师审核学生发起的申请 4. 老师通过申请，该课程加入学生的“我的课程”列表，页面重新渲染
分支流程	1. 填写申请理由时，如果拥有老师的选课码，可以直接输入选课码 2. 无需老师审核，学生自动加入该课程
异常流程	1. 输入错误的选课码 2. 系统提示选课码不正确
其他	无

在课程资料方面，每门课程都有文件库。老师可以对文件库进行文件上传、文件下载、以及文件分享操作，学生仅拥有浏览、下载、以及分享权限。老师上传课程资料的具体用例描述如表 3.3 所示。

表 3.3 上传课程资料的用例描述

ID	3
名称	上传课程资料
参与者	老师、助教
描述	老师和助教可以在课程文件库中上传课程资料
优先级	高
触发条件	用户点击上传文件按钮
前置条件	用户已登录，并鉴定为老师或助教身份
后置条件	无
正常流程	1. 用户点击上传文件按钮 2. 用户选择一个或多个本地文件，点击上传 3. 页面渲染文件上传进度 4. 上传成功，页面重新渲染文件库列表
分支流程	无
异常流程	1. 上传同名文件 2. 系统自动以增量的形式对文件进行重命名操作
其他	无

3.2.2 课程活动前端的功能性需求分析

图 3.3 是课程活动功能的用例图。在课程内部，老师可以发布基于某个特定主题的课程活动。系统内预置了四种类型的课程活动，包括考试、作业、项目活动、以及通知。老师、助教、学生是课程活动用例的主要参与者。

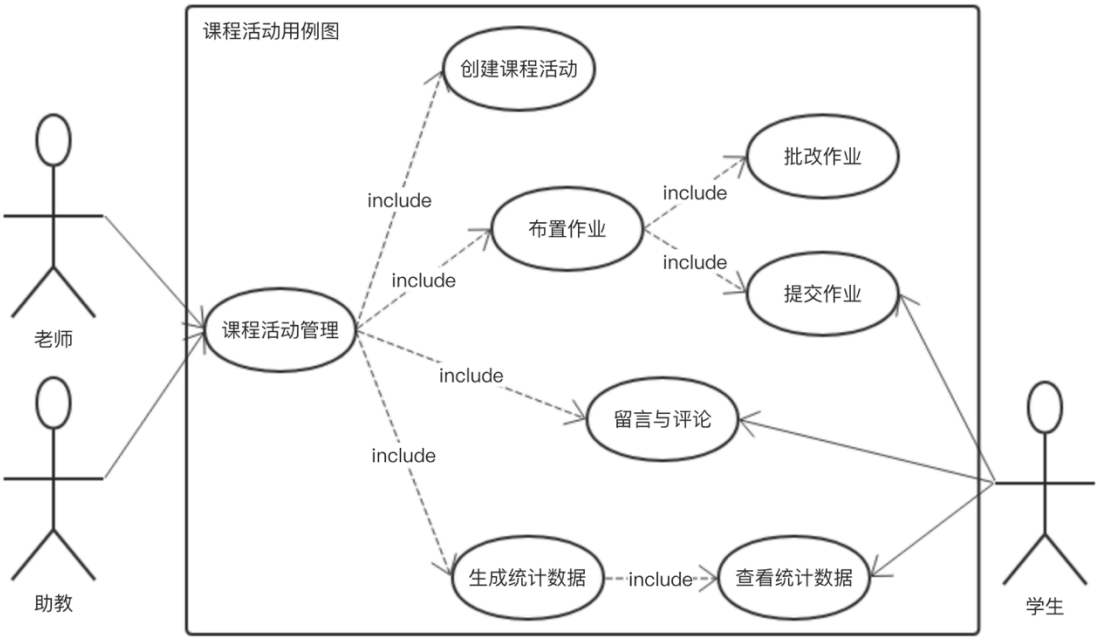

图 3.3 课程活动用例图

在创建课程活动时，系统提供富文本编辑区域以及附件上传功能。另外，对于作业一类的课程活动，老师还可以设置作业类型和作业截止时间。其中作业类型包括普通作业和小组作业两种。布置作业的用例描述如表 3.4 所示。

表 3.4 布置作业的用例描述

ID	4
名称	布置作业
参与者	老师
描述	老师可以在课程活动内布置作业
优先级	高
触发条件	用户点击创建课程活动按钮
前置条件	用户已登录，并鉴定为老师身份
后置条件	无
正常流程	1. 用户点击创建课程活动，选择类型为作业 2. 填写作业描述，上传作业附件 3. 选择作业类型，有个人作业和小组作业两种类型 4. 指定作业截止时间 5. 点击作业发布按钮，页面重新渲染课程活动列表
分支流程	1. 用户选择作业类型为小组作业 2. 系统将自动把作业推送各小组页面的活动列表中
异常流程	1. 用户选择过去的某一天为截止时间 2. 禁止用户操作
其他	作业截止时间默认为一周后的 23:59:59

课程活动详情页面提供评论和留言区域，老师、助教和学生都可以在此区域对活动内容进行讨论。助教可以在页面上下载到所有学生的作业并进行批改操作。批改完成之后，可以在页面上生成作业的统计数据，包括作业最高分、平均分、以及及格率等统计数据。统计数据可以帮助老师评估教学质量，改进教学方式，也可以帮助学生评估自己的作业完成情况，能够使得学生对自己的课程学习情况有一个更直观的认识。生成统计数据的用例描述如表 3.5 所示。

表 3.5 生成作业统计数据的用例描述

ID	5
名称	生成作业统计数据
参与者	助教、老师
描述	助教和老师在作业批改完之后，由系统自动生成作业统计数据
优先级	中
触发条件	用户点击生成作业统计
前置条件	用户已登录，并鉴定为助教或老师身份，作业已批改
后置条件	无
正常流程	1. 用户上传作业批改结果

	2. 用户选择统计类型，点击生成统计数据 3. 系统返回数据统计结果 4. 页面根据数据渲染统计图表
分支流程	1. 用户点击统计类型按钮 2. 默认所有类型全部选中，包括最高分、平均分、成绩分布
异常流程	1. 用户上传的批改结果格式不正确 2. 系统提示格式错误，请重新上传
其他	系统提供默认的批改结果登记表

3.2.3 小组管理前端的功能性需求分析

图 3.4 是小组管理功能的用例图。由于部分课程存在分小组进行课程活动的需求，所以课程内提供了相应的小组功能。老师、学生和助教都可以在课程内创建一个或多个小组。

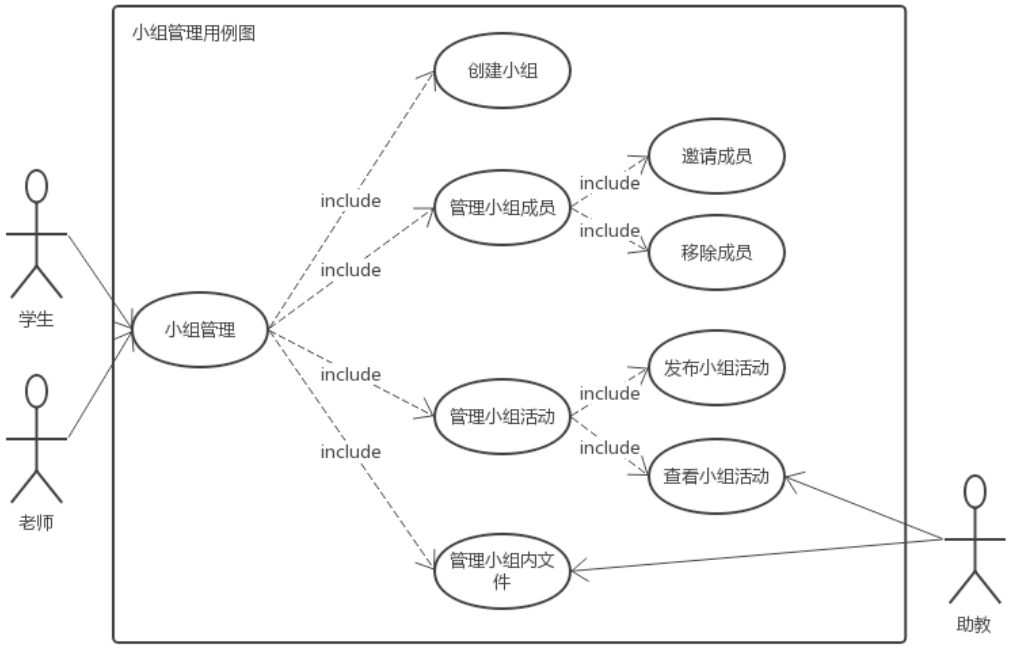

图 3.4 小组管理用例图

创建小组过程中，如果是学生创建，则该名学生自动成为小组的组长；如果是老师创建，则创建完成之后，老师所邀请进来的第一位学生将默认成为组长。创建小组的具体用例描述如表 3.6 所示。

表 3.6 创建小组的用例描述

ID	6
名称	创建小组

参与者	学生、老师
描述	课程内提供小组功能，学生和老师可以进行小组创建
优先级	高
触发条件	用户点击创建小组按钮
前置条件	用户已登录，并鉴定为学生或老师身份
后置条件	无
正常流程	1. 用户点击创建小组按钮 2. 填写小组名称，小组描述 3. 邀请小组成员 4. 提交创建表单 5. 页面重新渲染小组列表
分支流程	1. 老师邀请小组成员，浏览课程内人员列表 2. 被选中的第一个人员默认成为小组组长
异常流程	1. 小组名称与现有的小组重名 2. 系统提示小组名称已存在，提示用户进行修改
其他	学生创建小组时，默认创建者为组长

另外，每个小组都拥有与课程独立的小组活动功能和小组文件库功能。在小组活动中，小组内可以发布基于课程主题的活动内容，课程内类型为小组作业的课程活动会自动纳入到小组活动的列表中。这样做的目的是可以让小组各成员把注意力都集中在小组视角内，无需在课程视角和小组视角之间进行来回切换。老师和课程助教对每个小组的活动内容都有知情权，所以老师和助教可以进入课程内的所有小组进行观察跟踪。这样做能够使得小组活动透明化，老师能够更好地管理教学过程，助教也能更好地帮助学生进行问题解决。在小组文件库方面，每个小组成员都拥有对该文件库的全部读写操作，可以对文件或文件夹进行上传、下载、删除、以及分享等操作。小组内小组活动与文件库的功能可参考课程本身的课程活动与课程资料功能，两者需求类似，只是功能的使用范围不同。小组中的小组活动功能和文件库功能只关注小组本身，视角更集中。

3.2.4 课程会话前端的功能性需求分析

图 3.5 是课程会话的用例图。课程提供基于课程主题的会话功能。老师、学生、以及助教是会话功能的主要参与者。系统提供私聊和讨论组聊天两种会话方式，所有课程角色都可以创建讨论组发起群聊。讨论组创建者拥有讨论组成员管理的相关功能。

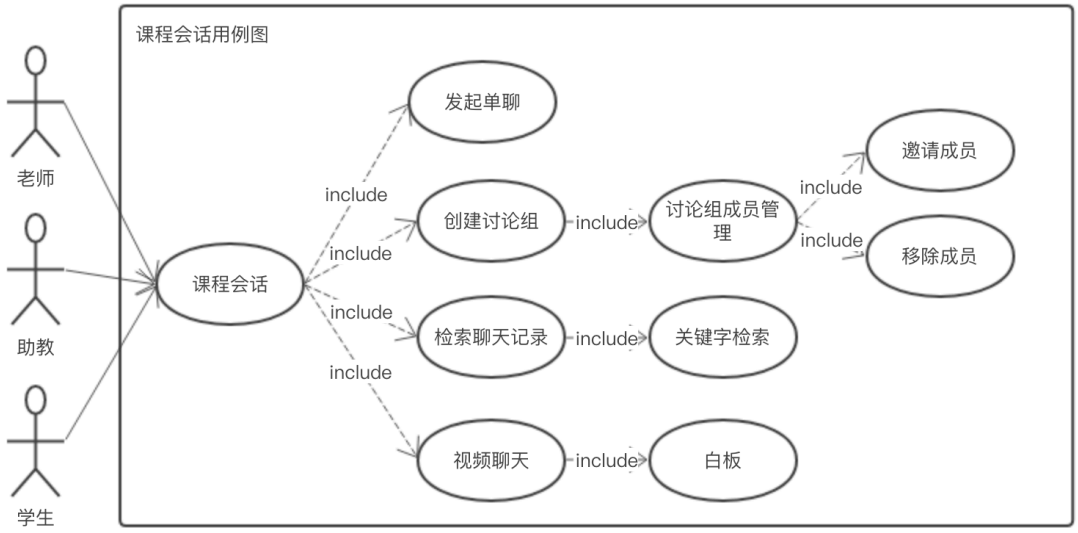

图 3.5 课程会话用例图

系统内所有会话的聊天记录都会被保存，会话参与者可以在系统上查询不同类型的聊天记录，包括图片、文件、普通文本等。用户也可以根据时间范围进行聊天记录的筛选。检索历史聊天记录的具体用例描述如表 3.7 所示。

表 3.7 检索聊天记录的用例描述

ID	7
名称	检索聊天记录
参与者	教务员、老师、助教、学生
描述	课程内提供会话功能，用户可以在讨论组和单聊内检索历史记录
优先级	中
触发条件	用户点击查看聊天记录按钮
前置条件	用户已登录，且已发起过会话
后置条件	无
正常流程	1. 用户点击查看聊天记录按钮 2. 用户选择检索类型，包括文件、图片、时间范围 3. 用户输入检索关键字 4. 页面显示最新的 25 条聊天记录 5. 用户点击查看更多 6. 页面显示后 25 条，直到所有聊天记录都已呈现
分支流程	1. 用户选择检索类型为时间范围 2. 用户选择某一个时间范围 3. 页面显示该时间范围内前 25 条所有类型的聊天记录
异常流程	1. 用户已被移出讨论组 2. 用户只可检索被移出前的聊天记录
其他	关键字搜索同时匹配文件名和文本内容

另外，系统还提供了视频聊天的相关功能。在视频聊天的过程中，页面上会呈现一块用于涂画的白板。白板支持画笔、橡皮擦、形状、图形插入、画面缩放

等常见操作。参与聊天的所有成员都拥有白板的读写操作。视频聊天的具体方式如表 3.8 所示。

表 3.8 视频聊天的用例描述

ID	8
名称	视频聊天
参与者	教务员、老师、助教、学生
描述	用户可以在会话功能中进行视频聊天
优先级	低
触发条件	用户点击视频聊天按钮
前置条件	用户已登录，且已发起过会话
后置条件	无
正常流程	1. 用户点击视频聊天按钮，发起视频聊天 2. 传输视频图像画面 3. 提供白板操作区域 4. 用户在白板中进行绘画操作 5. 结束视频聊天
分支流程	1. 在讨论组内发起视频聊天 2. 画面显示当前发言人的视频图像
异常流程	1. 用户掉线 2. 终止视频聊天
其他	视频聊天过程提供录制功能

3.2.5 知识体系前端的功能性需求分析

图 3.6 是知识体系功能的用例图。除了基本的教学活动支持以外，系统还提供了知识体系构建的相关功能。老师和学生是知识体系功能的主要参与者。

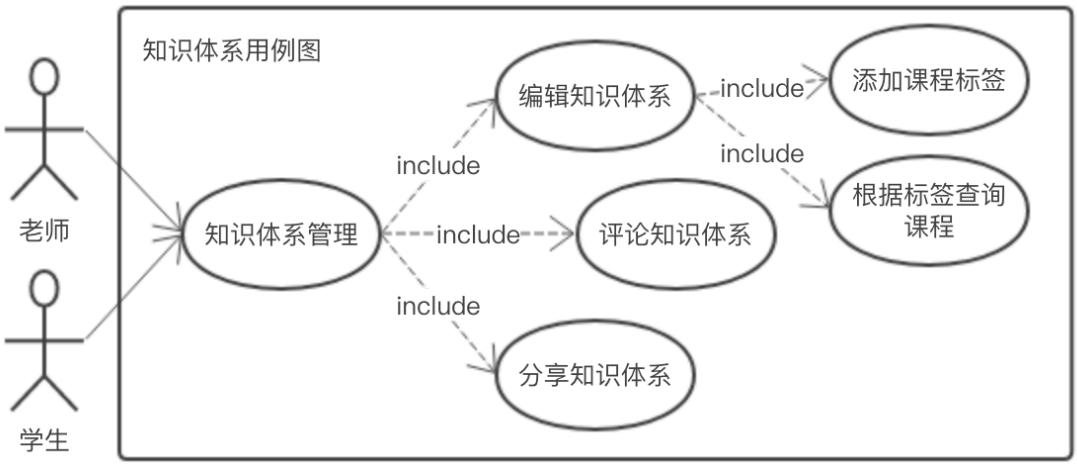

图 3.6 知识体系用例图

在传统的教学支持系统中，学生对课程知识的获取相对碎片化，每个学生的

知识领域都是一个信息孤岛，无法对各领域的知识进行归纳总结，形成严谨有序的知识体系。另外，学生之间的经验分享也相对闭塞。在超级账号教学支持系统中，学生和老师可以自由创建自己的知识体系。用户可以对其他用户的知识体系进行评论，也可以将自己的体系分享给他人。知识体系是一个树状结构，由多个标签组成。老师可以给自己的课程添加相应标签，学生可以在知识体系中根据标签查询到所有与之相关的课程列表，从而进行更有针对性的学习，使得自己的知识体系更加完整。编辑知识体系的具体用例描述如表 3.9 所示。

表 3.9 编辑知识体系的用例描述

ID	9
名称	编辑知识体系
参与者	老师、学生
描述	用户可以对自己的知识体系进行编辑
优先级	高
触发条件	用户点击编辑知识体系按钮
前置条件	用户已登录，且身份鉴定为老师或学生
后置条件	无
正常流程	1. 用户点击编辑知识体系按钮 2. 根据标签搜索相关课程 3. 将课程加入到知识体系的相应标签下 4. 保存编辑
分支流程	无
异常流程	1. 与标签相关的课程搜索结果为空 2. 系统提示暂无相关课程
其他	无

3.3 超级账号教学支持系统前端的非功能性需求分析

课程内的会话采用 WebSocket 进行消息传输。为了保证会话功能的可用性，系统采用断线重连机制进行自动重连。当页面监听到用户已掉线的事件后，会自动向服务端发送重连请求。如果当前连接数量过多，系统会暂时中断连接不活跃的用户，以保证当前活跃用户的功能可用性。采用这种机制能够保证 99.99%的会话功能可用性。

页面在安全性方面需防范跨站请求伪造的攻击。页面在提交 HTTP 请求时，所有的敏感字段不应该放在 Request Body 中，而需要放在 Request Header 中

进行传递。这样能够保证异步请求在使用 HTTPS 协议的情况下无法进行 Request Header 的伪造。

页面须保证易用性，避免频繁打断用户操作，应给用户提供流畅的操作体验。当页面有数据需要更新时，系统不会刷新整个页面，只会对部分组件进行重新渲染。且在等待过程提供加载动画，降低操作延迟感知。

页面性能应该满足用户在提交表单请求后 3 秒内响应用户请求。如果由于网络中断原因导致请求发送失败或者响应接收失败，应在页面给出相关提示，页面的最长响应时间不应该超过 10 秒。

3.4 本章小结

本章主要对超级账号教学支持系统前端的需求进行了分析。本章首先对系统需求进行了整体概述，详细描述了该系统与它的母产品“超级账号”之间的关系，解释了该系统使用事务管理模型来进行设计的原因和优势。接着，本章对该系统的功能性需求进行了详细的分析。系统功能性需求主要分为五大块：课程管理功能、课程活动功能、小组管理功能、课程会话功能、以及知识体系功能。最后本章对超级账号教学支持系统前端的非功能性需求进行了介绍，主要阐述了系统的可用性、安全性、易用性和性能这四个方面。

第四章 超级账号教学支持系统前端的设计

4.1 超级账号教学支持系统前端的概要设计

图 4.1 是超级账号教学支持系统前端的总体架构图。系统前端的架构采用经典的 MVC 模式进行设计。View 层是页面展示层，Controller 层是核心逻辑处理层，Model 层维护了系统前端的所有页面状态和从服务端拉取过来的数据。

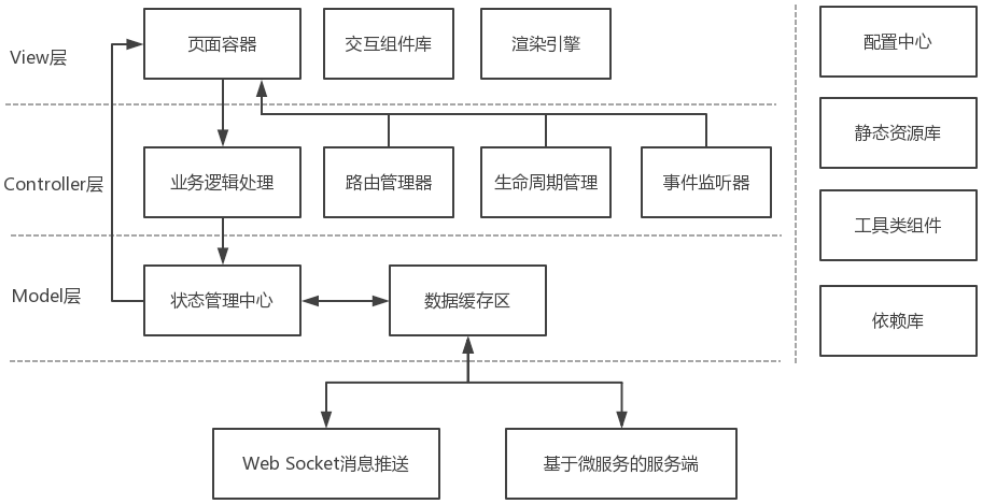

图 4.1 超级账号教学支持系统前端的总体架构图

具体来说，View 层负责完成页面内容展示以及用户交互的相关工作，所有的页面容器通过渲染引擎渲染成 HTML 页面输出到浏览器中，另外前端通过 Ant-Design 提供的丰富的组件库来完成页面交互工作。

Controller 层负责完成业务逻辑处理以及页面容器的管理工作。在业务逻辑处理方面，页面上的每一个数据更新请求都会向 Redux 分发一个 action 动作，action 会向 Model 层的状态管理中心请求最新数据，最后由状态管理中心直接把最新数据返回给页面，页面再进行重新渲染。除此之外，Controller 层还完成了页面路由管理、页面组件生命周期管理、以及页面事件监听的工作。路由管理主要由 React-Router 实现；生命周期方面，页面上的每个组件都有 7 个生命周期，它们可以灵活地控制组件从创建到更新、再到销毁的完整过程；另外，页面

上通常会绑定很多的事件监听器，这些事件的响应过程也由 Controller 层负责。

Model 层是系统的数据管理层，所有从服务器拉取的数据都会在 Model 层进行存储。它主要由状态管理中心和数据缓存区两个部分组成。在系统中，状态分为组件自身状态以及多个组件之间共享的状态。组件自身状态由页面容器进行独立控制，状态管理中心主要用来控制多个组件之间进行共享的状态。这样可以方便地保证数据在多个组件之间的一致性，也能够让组件更方便地进行数据传递。数据缓存区会对大型数据进行浏览器端的缓存工作，一些实时性要求不高的数据都会缓存在其中。这样可以避免向服务器频繁地发送请求，降低服务器压力。

在 View 层，Controller 层以及 Model 层以外，系统内还有一些辅助型模块对 MVC 层进行支撑。配置中心主要用来管理系统的 api 配置，包括服务器 ip 的定义、端口的定义等。静态资源库主要包含了页面图标、图片、字体等资源。工具类组件模块提供了系统所必须的一些处理工具，包括时间语义化工具、与 oss 服务器进行数据交换的工具等。依赖库包含了项目使用到的所有第三方库。

最后，整个前端通过 WebSocket 和 Http Request 与后端进行数据交换，系统的后端包括了负责会话功能的 Node.js 层，以及基于微服务的服务端两部分。

图 4.2 是系统前端的功能模块划分图。整个前端主要由五大块组成，分别为课程管理模块、课程活动模块、会话模块、小组模块以及知识体系模块。课程管理模块由课程信息管理子模块、课程资料管理子模块、以及课程人员管理子模块组成。小组作为课程的一个子集，也拥有三个类似的子模块，分别为小组活动管理子模块、小组资料管理子模块、以及小组人员管理子模块。

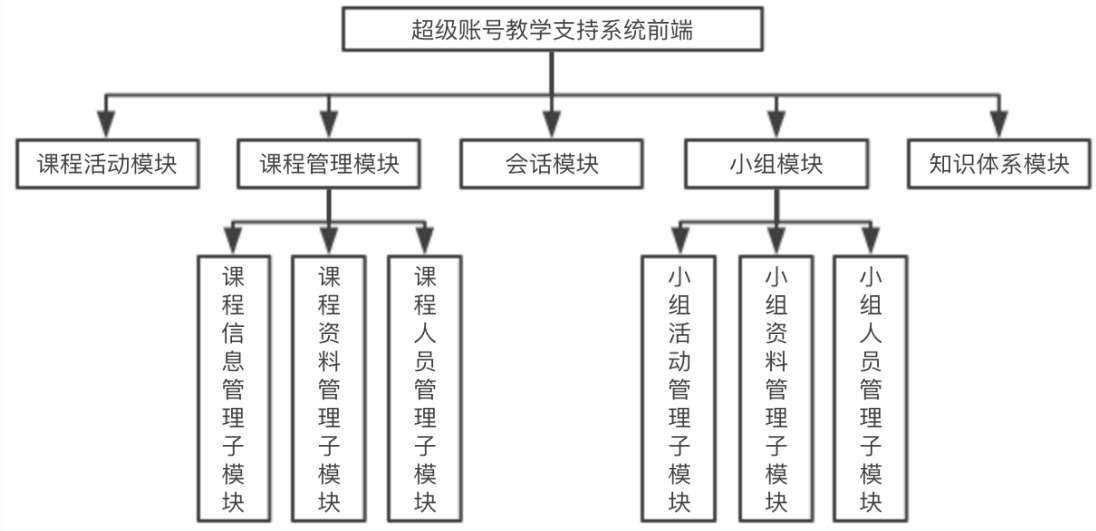

图 4.2 功能模块划分图

4.2 课程管理模块前端的设计

4.2.1 课程信息管理子模块前端的设计

课程管理模块主要由课程信息管理子模块、课程人员管理子模块、以及课程资料管理子模块三个部分组成。图 4.3 是课程信息管理子模块的时序图。该子模块主要实现了课程创建、课程信息修改的相关功能。

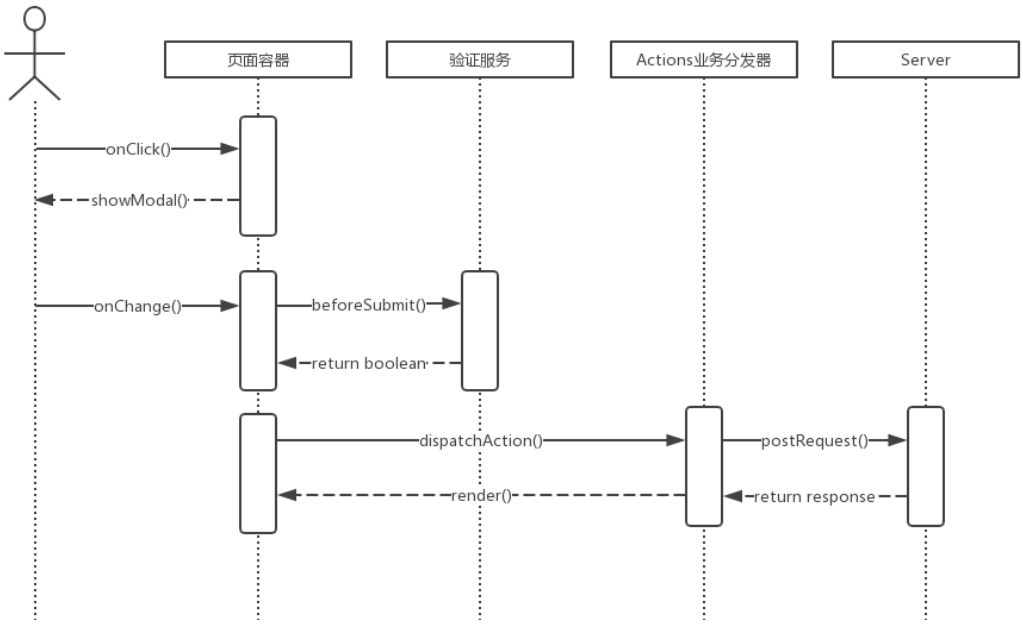

图 4.3 课程信息管理时序图

从图中可以看出，用户在页面点击创建课程或修改课程信息的按钮之后，页面容器触发 onClick()事件，弹出课程信息表单对话框。用户对表单信息进行填写之后，点击对话框提交按钮，触发 beforeSumit()事件，之后组件验证服务会对表单内容进行合法性验证，包括表单字段是否有空白字段、字段长度是否在规定长度内等。如果输入不合法，表单组件会提示用户进行修改；如果合法，页面容器接着会触发 dispatchAction()动作，将表单内容提交给 Actions 逻辑处理中心。之后，action 会将表单内容包装成 Http Request 发送给后端服务器，服务器返回提交结果。如果表单提交成功，action 会触发页面容器的 render()动作，最后，页面容器会对课程列表组件进行部分重新渲染，将最新的课程信息数据显

示在页面上。

图 4.4 是课程信息管理的类图。CourseIndexContainer 是课程信息的主容器，它继承了 React 的 Component 类，由 CourseInfoContainer 和 CourseListContianer 两个子容器组成。前者用来渲染课程的详细信息，后者用来渲染课程列表。课程列表容器又由 CourseCard 和 CreateCourseModal 这两个组件组成。前者负责渲染列表中的每一门课程卡片，后者是创建课程的对话框。CourseAction 是课程信息的业务逻辑处理类，CourseInfoContainer、CourseListContainer、以及 CreateCourseModal 都依赖于它。CourseAction 类包含了获取课程列表、创建新课程、以及获取课程详情等业务方法。CourseReducer 是课程信息的数据中心类，负责存储所有与课程信息相关的容器和组件的数据。

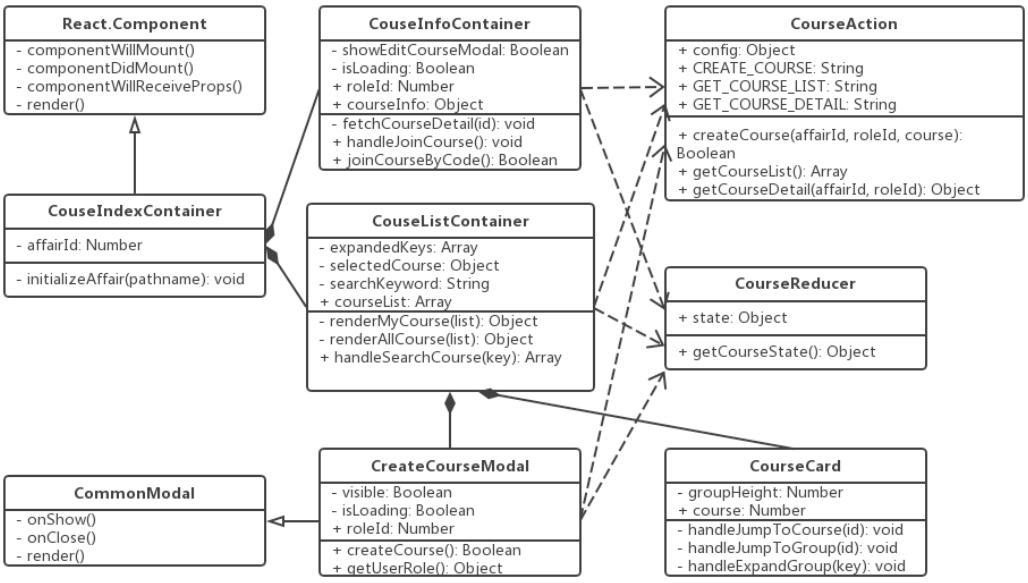

图 4.4 课程信息管理类图

4.2.2 课程人员管理子模块前端的设计

图 4.5 是人员管理时序图。每一门课程的人员管理主要包括老师管理和助教管理两个部分。用户进入人员管理界面点击管理按钮之后，首先选择管理动作类型。如果是进行人员邀请，页面会渲染出系统所有可选的人员列表。在渲染人员列表的过程中，如果前端数据 Cache 已有列表的缓存数据，会优先从 Cache 读

取数据；如果 Cache 没有数据，会从远端服务器拉取数据，拉取得到的数据会写入 Cache。之后，用户即可从人员列表中勾选相关人员，勾选完成之后触发组件的 onSubmit()动作，页面容器分发一个表单提交 action 给服务器，服务器返回提交结果。如果是移除人员，用户选择要移除的人员之后，页面容器会弹出气泡确认框进行二次确认。如果确认移除，则提交移除表单。

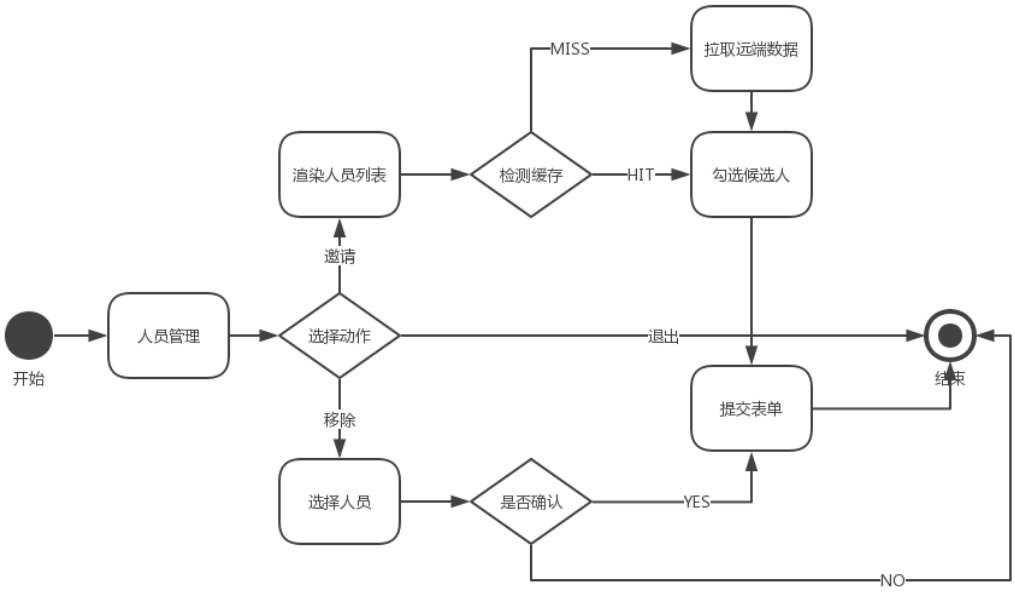

图 4.5 人员管理流程图

图 4.6 是课程人员管理的类图。MemberContainer 是人员管理的主容器，由 InviteMemberModal 和 MemberCard 两个子组件组成。前者继承 CommonModal 类，是邀请课程任课老师、助教的对话框。MemberCard 是课程内每个人员的角色卡片，卡片上展示了角色的基本信息，包括 id、姓名、性别、以及联系方式等。MemberAction 是业务逻辑处理类，MemberContainer 和 InviteMemberModal 依赖于 MemberAction 进行业务交互。MemberReducer 是人员管理的数据中心。

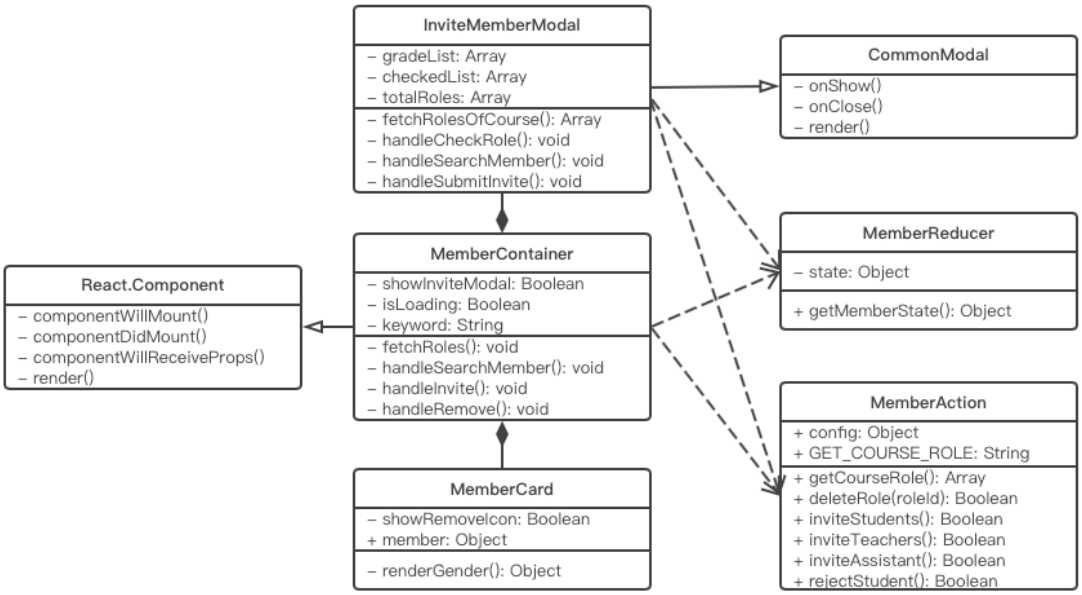

图 4.6 人员管理类图

4.2.3 课程资料管理子模块前端的设计

图 4.7 是课程资料管理时序图。资料下载、资料删除、以及资料页面预览的逻辑设计相对简单，此处重点描述用户进行资料上传的设计逻辑。

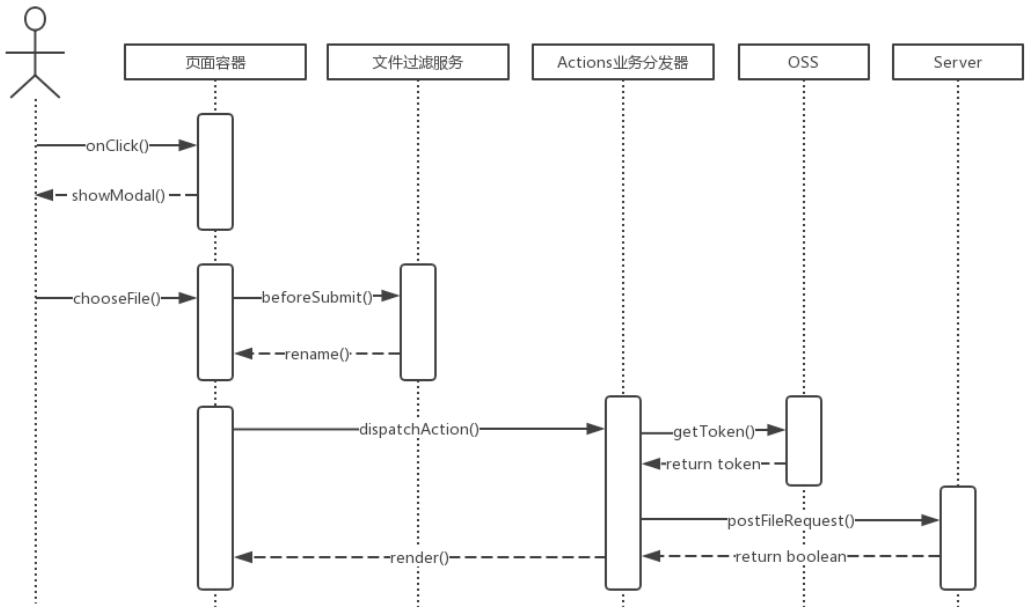

图 4.7 课程资料管理时序图

用户点击上传文件按钮之后，页面会弹出文件上传对话框，用户选择完需要上传的文件，页面容器会触发 FileRenameService 中的 rename()方法，对文件

名进行过滤处理。如果文件名和已有文件重复，会自动给文件名加上偏移增量。重命名完成之后，页面容器会分发一个 action 给 Actions 业务逻辑处理器，action 会先从 OSS 服务器获取文件上传 token，token 定义了文件在 oss 服务器的存储位置。action 拿到 token 之后，将 token 内容封装成 Http Request 发送给后端服务器，服务器返回 token 存储结果。如果文件上传成功，action 会触发页面容器的 render()方法，容器对文件列表组件进行重新渲染。

图 4.8 是课程资料管理类图。FileListContainer 是课程资料的主容器，负责展示课程资料的文件列表，依赖于 FileAction 类为该容器提供业务逻辑处理方法。它依赖于三个子组件，分别是 FilePreviewModal、UploadFileModal、以及 CreateFolderModal。第一个负责完成文件预览，第二个用于文件上传，第三个用于新建文件夹。OSS 类是负责文件上传至 OSS 服务器的业务逻辑处理类。上述容器和组件都依赖这个类，这些容器或组件会首先与 OSS 类进行交互，再与 FileAction 进行交互。FileReducer 负责完成课程资料的文件缓存工作。

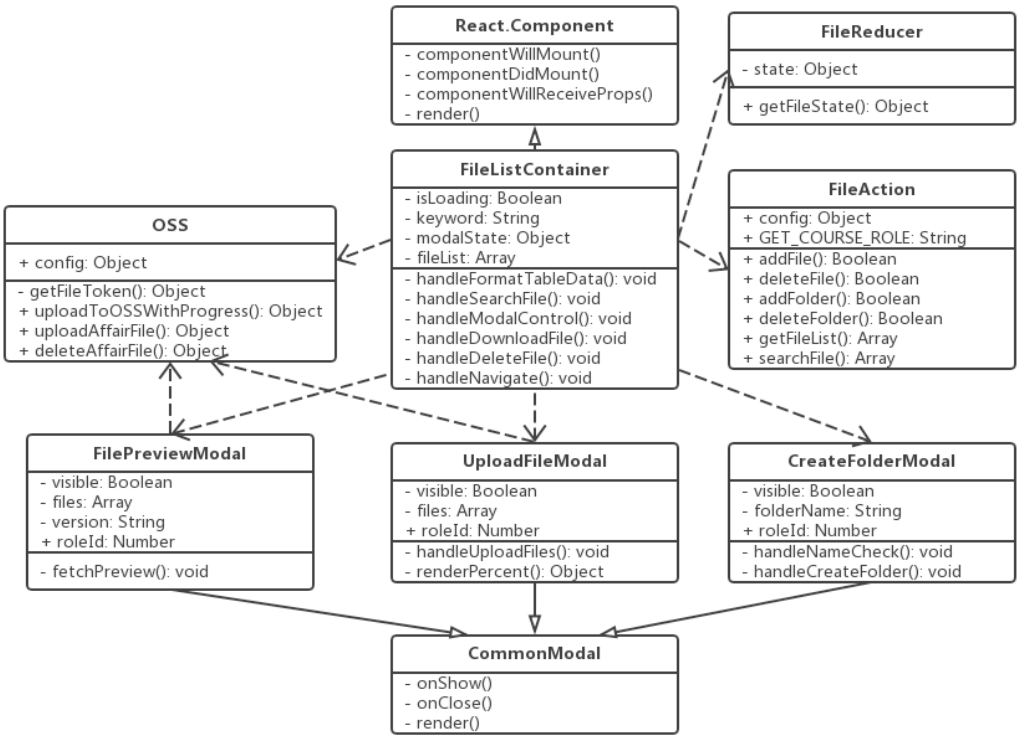

图 4.8 课程资料管理类图

4.3 课程活动模块前端的设计

图 4.9 是课程活动管理的时序图。在获取课程活动列表的逻辑设计上，用户进入活动页面后会触发页面容器的 getList()动作，页面容器首先会访问前端数据 Cache。如果 Cache 中缓存过活动列表，会直接将数据返回给页面容器进行组件渲染；如果 Cache 中缓存失效，会触发 dispatchAction()动作。Actions 业务分发器接到 action 之后，向 Server 发送列表 getRequst()，Server 返回数据之后，action 会触发 updateCache()方法进行缓存刷新。在创建课程活动的逻辑设计上，用户点击创建按钮触发 createActivity()动作，页面容器分发 action，action 将活动表单封装成 Http Request 发送给 Server，Server 返回创建结果。如果创建成功，action 会将新建的课程活动写入 Cache 中的活动列表进行缓存刷新。最后返回给页面容器，触发 render()方法，页对活动列表组件进行重新渲染。

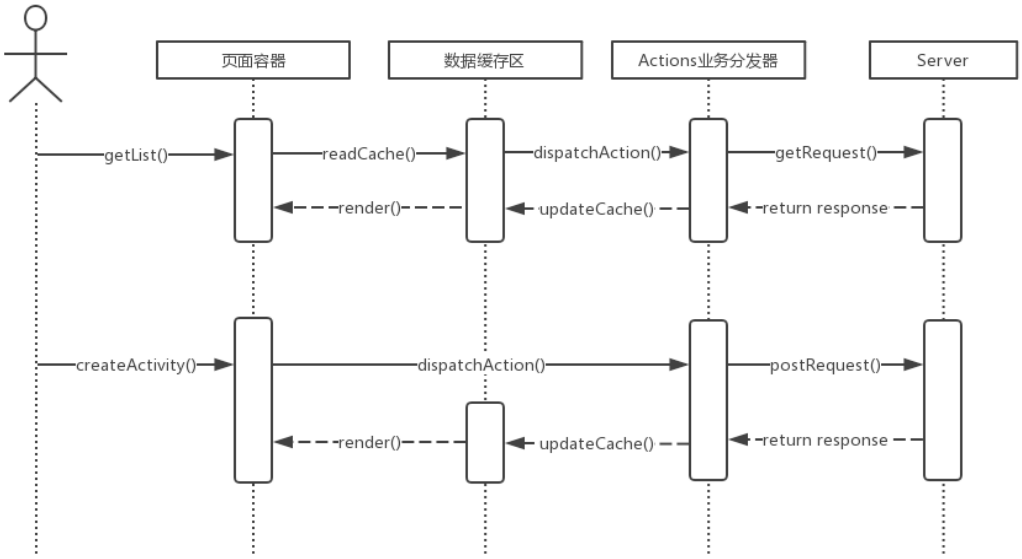

图 4.9 课程活动管理时序图

图 4.10 是课程活动管理的类图。ActivityListContainer 展示课程活动列表，由三个子组件组成：CreateActivityModal 是创建课程活动的对话框；ActivityItem 负责展示活动列表中每一项的数据渲染；ActivityDetailContainer 用来展示活动详情，它又依赖于 HomeworkContainer 以及 CommentContainer 两个子组件，前者负责活动作业的内容渲染，后者负责渲染活动评论列表。ActivityAction 负责完成上述所有容器或组件的业务逻辑处理工作，所有页面上派发的 action 都

会集中到这个类进行处理。ActivityReducer 负责完成活动列表的缓存。

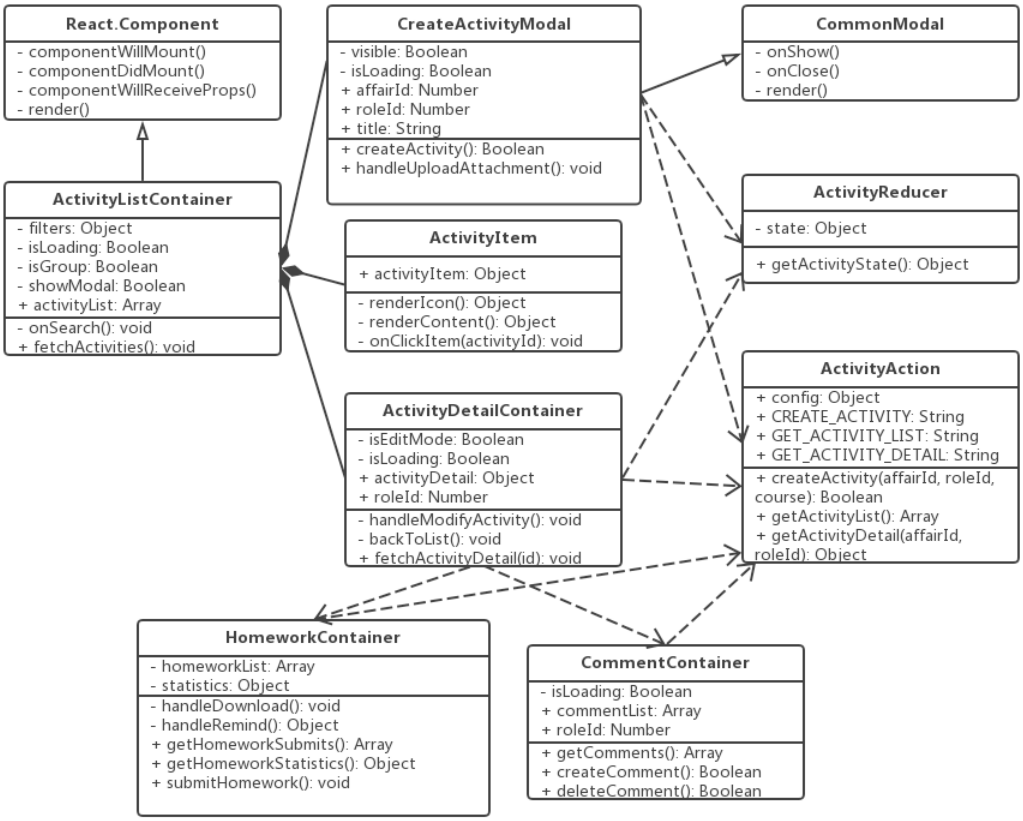

图 4.10 课程活动管理类图

4.4 会话模块前端的设计

图 4.11 是会话模块的时序图。在操作逻辑上，发送消息与接收消息类似，查看历史聊天记录与创建讨论组类似，图中重点描述了发送消息和查看历史聊天记录的时序逻辑。用户在文本组件输入消息内容后，触发页面容器的 onSend()方法，页面容器会对消息内容进行封装，加上用户参数和课程相关参数，打包后的消息体会发送给前端的 ChatClient SDK。SDK 主要封装了消息转发的核心逻辑。SDK 在收到消息体后，会将它转发给置于 Node.js 层的 WebSocket Server，之后，WebSocket Server 会将消息广播给所有消息体的发送对象。广播信息会被 SDK 拦截到，对消息进行解析之后，返回给页面容器，页面容器最终触发 renderMsg()方法，将消息显示在聊天面板组件上。查看聊天历史记录方面，用户点击查询按钮之后，触发页面容器的 showHistory()动作。由于聊天历史记录

的实时性要求很高，所以聊天记录不在在 Cache 中进行缓存，页面容器会直接分发一个 action 向 WebSocket Server 进行数据拉取。拉取数据采取分页查询。action 接收到 Server 的响应之后，将数据返回给页面容器，触发 render()动作，对聊天记录组件进行重新渲染。

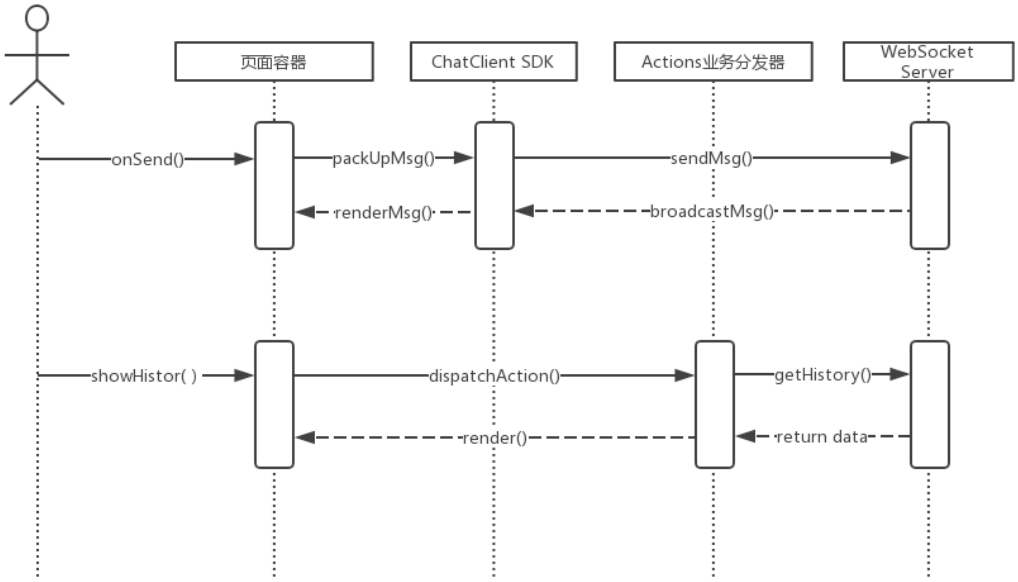

图 4.11 会话模块时序图

图 4.12 是会话模块的类图。课程会话由单独的 ChatContainer 主容器进行页面渲染。它由五个子组件组成：ListPanel 负责渲染会话列表以及通讯录；MessagePanel 负责渲染聊天主界面，所有的对话信息都会渲染在这个面板里面。另外，MessagePanel 依赖于 ChatBox 组件，该组件是聊天的富文本输入框，输入方式包括文字、表情、图片、以及文件等，是一个公用组件；MemberPanel 负责渲染群聊的成员列表，在里面可以对成员进行邀请或移除操作；ChatRecordPanel 负责渲染聊天记录，可以对文字、图片、或者文件类型的聊天记录进行分页检索；CreateGroupChatModal 是创建群聊讨论组的对话框。在 ChatContainer 类中引入了 Client 变量，该变量可以被所有的子组件使用，其内部封装了接发消息的相关业务逻辑，负责与 WebSocket SDK 进行交互。ChatAction 负责完成主容器与服务端的业务交互工作。ChatReducer 是数据管理中心，包括最近会话列表在内的数据都会在其中进行缓存。

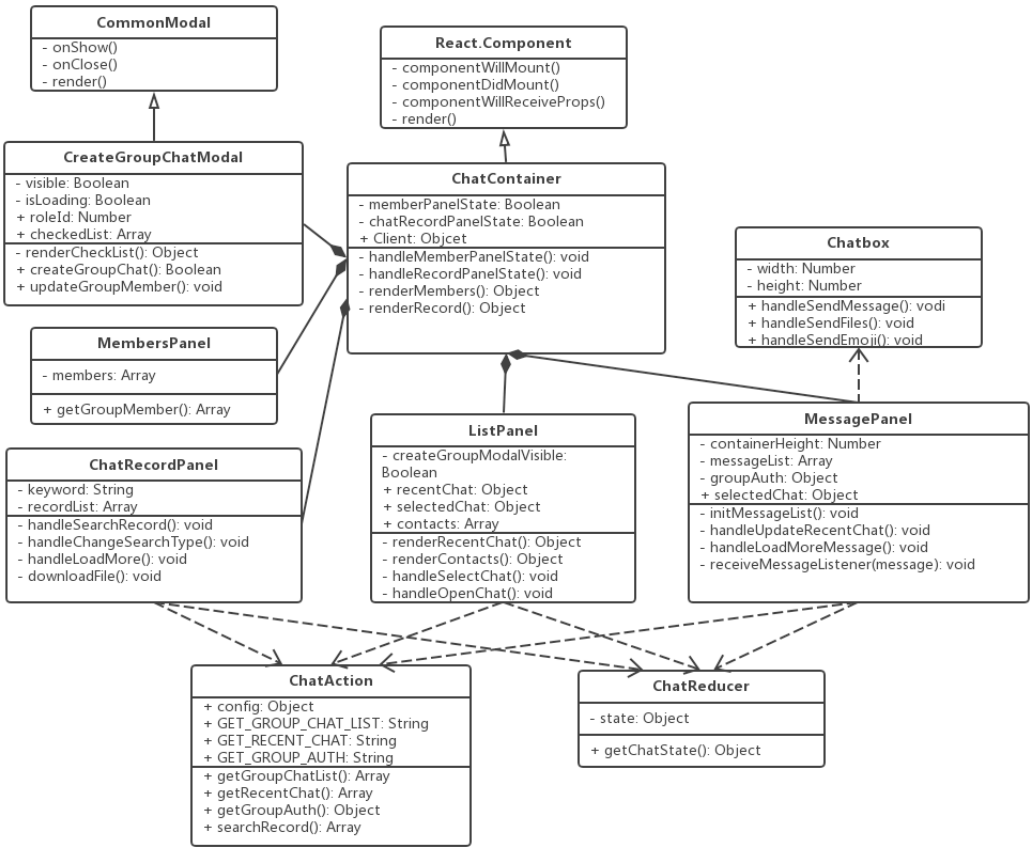

图 4.12 会话模块类图

4.5 小组模块前端的设计

图 4.13 是小组模块流程图。创建小组时，页面首先会检测创建者的角色类型。如果当前角色是学生，则该名学生自动成为小组组长；如果当前角色是老师，老师需要在接下来的人员列表中勾选一名学生作为组长。学生和老师在对小组进行人员邀请时，页面会渲染出当前课程内的所有人员列表。人员列表数据如果在 Cache 中缓存有效，会直接进行渲染，用户即可在列表中进行人员勾选。勾选过程中，页面会提示老师勾选的第一名学生将成为这个小组的组长。勾选完成之后，提交人员名单给 Server，Server 返回小组创建结果；如果缓存失效，会首先向服务器请求数据，再进行之后的勾选动作。小组内小组活动的设计逻辑与课程活动类似，小组资料的设计逻辑与课程资料类似，可参考 4.2 和 4.3 节。

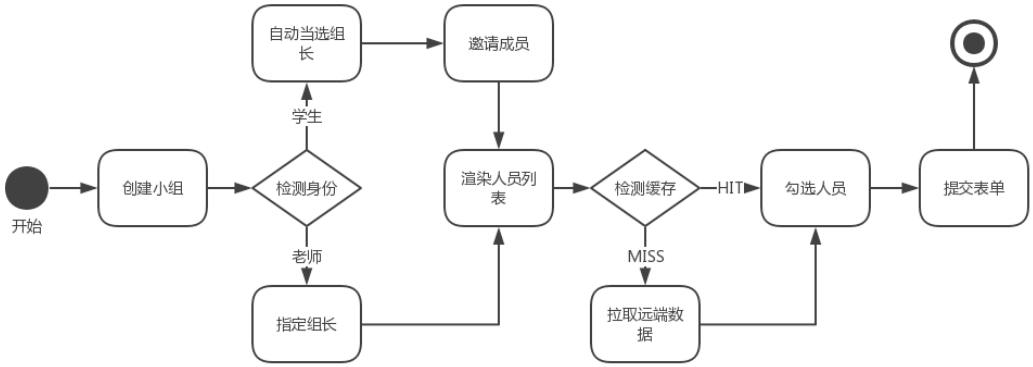

图 4.13 小组模块流程图

图 4.14 是小组模块类图。GroupContainer 是小组的主容器，由四个子组件组成：GroupCard 是每个小组的卡片信息；ApplyToJoinModal 是申请加入小组的对话框，负责处理加入小组的相关页面逻辑；CreateGroupModal 是创建小组的对话框，负责渲染创建小组表单；InviteMemberModal 是邀请小组成员对话框，负责完成成员邀请的相关页面逻辑。GroupAction 负责完成所有与小组管理相关的业务逻辑，包括与 GroupReducer 数据中心的交互以及与远端服务器的交互逻辑。GroupReducer 负责缓存所有的小组列表。

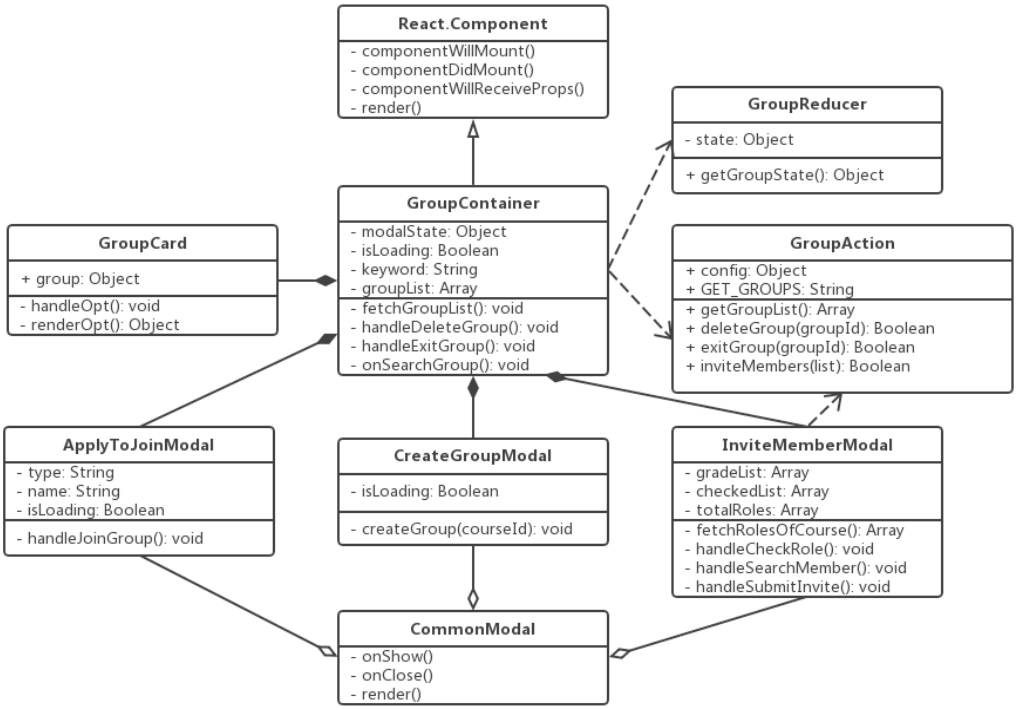

图 4.14 小组模块类图

4.6 知识体系模块前端的设计

图 4.15 是知识体系模块的流程图。知识体系是一个树状结构，每个树节点都是一个知识标签。每个知识标签都与一门或多门课程关联，用户可以点击标签查看与之相关的课程主页。用户在创建完知识体系之后，可以对知识体系进行变更操作，分别有添加标签、删除标签、以及移动标签三种操作。在删除标签时，页面会对用户操作进行二次确认。添加标签时，页面会弹出对话框，让用户输入标签内容，并勾选与标签相关的课程。操作完之后，页面会对知识体系进行重新渲染。之后，用户可以对知识体系进行评论和分享等互动操作。

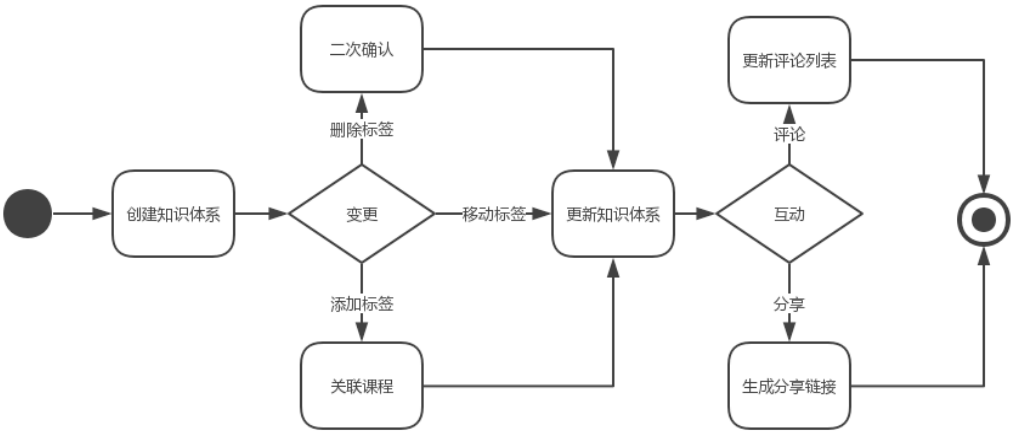

图 4.15 知识体系模块流程图

图 4.16 是知识体系的类图。KnowledgeContainer 负责渲染知识体系的树形图，以及知识体系的评论列表。KnowledgeContainer 依赖于 AddTagModal 子组件以及 KnowledgeSharePage 子组件。前者负责完成在知识体系内添加标签的业务逻辑，后者负责渲染知识体系分享页面。KnowledgeAction 负责完成所有与知识体系相关的业务逻辑操作。KnowledgeReducer 是知识体系数据中心，知识体系树状结构会在数据中心进行缓存。

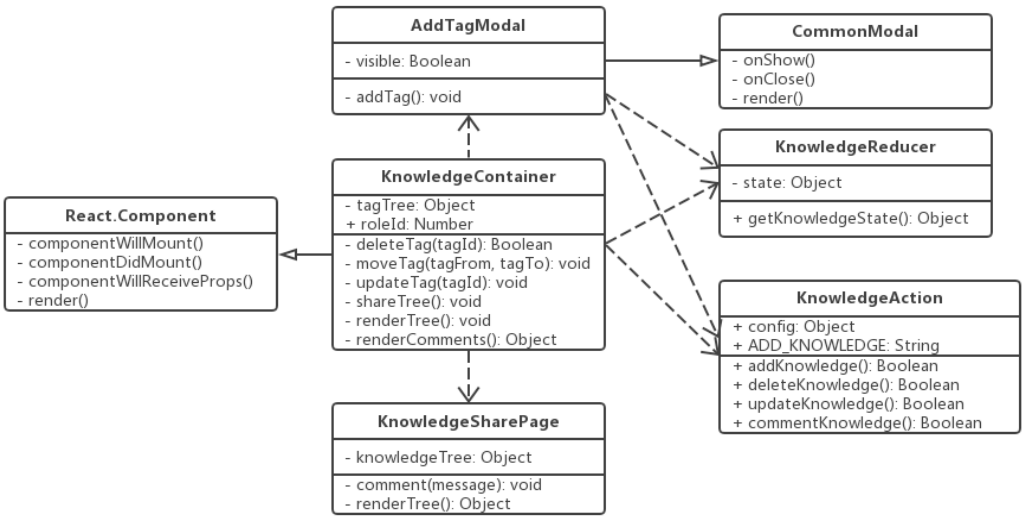

图 4.16 知识体系模块类图

4.7 本章小结

本章主要阐述了超级账号教学支持系统前端的总体设计。首先介绍了系统前端的概要设计，并给出了架构设计图和功能模块划分图；接着本章将系统前端分解为五个大块并逐个进行阐述，包括课程管理模块、课程活动模块、会话模块、小组模块、以及知识体系模块，并给出了相应的时序图和类图。

第五章 超级账号教学支持系统前端的实现

5.1 实现环境概述

超级账号教学支持系统前端是采用 MVC 模式所设计的单页面应用。系统基于 React16.2.0、Redux3.7.2、React-Router4.2、AntDesign2.13.11、以及其他 React 体系内的相关技术进行组件化开发。所有容器和组件采用 jsx 语法编写页面布局以及页面逻辑，采用 Sass 语言进行页面样式编写。

项目最终通过 Babel6.26.0 完成 JavaScript 的解析编译工作，将 ES6 或更高版本的语法转化成 ES5 语法，然后通过 Webpack2.5.1 进行构建打包。项目同时提供开发模式和生产模式两种运行环境，在开发模式下，Webpack 会把项目文件打包成一个 main.js 文件以及多个 chunk[hash].js 文件，chunk 文件根据页面路由进行按需加载，Webpack 通过 hot-reload 的方式对不同 chunk 进行代码热更新。在生产模式下，项目首先通过 npm install 指令完成依赖安装，再通过 npm run build 指令完成构建，最终 Webpack 会把项目打包成 entry.js、vendor.js、以及多个 chunk[hash].js 文件。entry.js 是项目的入口文件；vendor.js 是所有核心依赖库打包合并后的文件，且该文件版本稳定，可以在浏览器进行长期缓存；chunk[hash].js 是不同模块打包后的文件，后期对不同模块的修改，浏览器只需更新对应的 chunk 文件即可。

Webpack 构建完成之后，项目会在 Node 层通过 Express 启动一个 Web 服务器作为运行环境。在部署方式上，项目最终通过 docker 打成镜像，然后部署在云服务器，由 nginx 对 Express 服务器进行反向代理。

5.2 课程资料模块前端的实现

5.2.1 文件批量上传的实现

图 5.1 是文件进行批量上传的代码。inputFiles 变量指向用户选择的文件列

表，在上传之前，会对 inputFiles 内的文件进行重名检查。检查时，通过小数点分隔符将文件名与文件后缀分开，然后通过正则表达式，匹配现有的文件库列表中，是否有重名文件存在。如果存在，则自动在文件名的后面添加增量。正则表达式通过 escape()方法对特殊字符进行了逃逸处理，避免特殊字符对正则表达式的影响。在检查完所有的文件之后，页面会渲染出文件上传对话框显示文件上传状态，通过参数传递的方式，将 inputFiles 传递给对话框，并将对话框的显示状态 UPLOAD_FILE_MODAL_STATE 设置为 true，多个文件会同时并行上传。待文件上传完成之后，关闭对话框。

```
handleReadyToUploadFiles = (inputFiles, targetFolderId, isUploadFolder) => {
  const { path } = this.props
  // 检查重名
  inputFiles.forEach(f => {
    let nth = 0
    let newFile = f.name.split('.')
    // 文件名中含有多个 . 的情况
    if (newFile.length > 2) {
      newFile[0] = newFile.slice(0, -1).join('.')
      newFile[1] = newFile[newFile.length - 1]
    }
    this.state.fileMap.getIn([path, 'files']).forEach(v => {
      const existFile = v.get('name').split('.')
      let existFileName = escape(existFile[0])
      if (existFile.length > 2) {
        existFileName = existFile.slice(0, -1).join('.')
      }
      const regex = new RegExp('\^' + escape(newFile[0]) + '(%28)?([0-9]*)(%29)?\$')
      const regexRes = regex.exec(existFileName)
      if (regexRes && +regexRes[2] >= nth) {
        nth = (+regexRes[2]) + 1
      }
    })
    if (nth) {
      f.fileName = newFile[0] + "(" + nth + ")." + newFile[1]
    }
  })
  this.setState({
    readyToUploadFiles: inputFiles
  })
}
```

图 5.1 文件批量上传的代码示例

图 5.2 是文件批量上传的界面截图。点击界面右边上传按钮触发文件上传操作，用户可选择多个文件同时上传。上传对话框中会渲染每个文件的上传进度，上传完成后可关闭对话框。

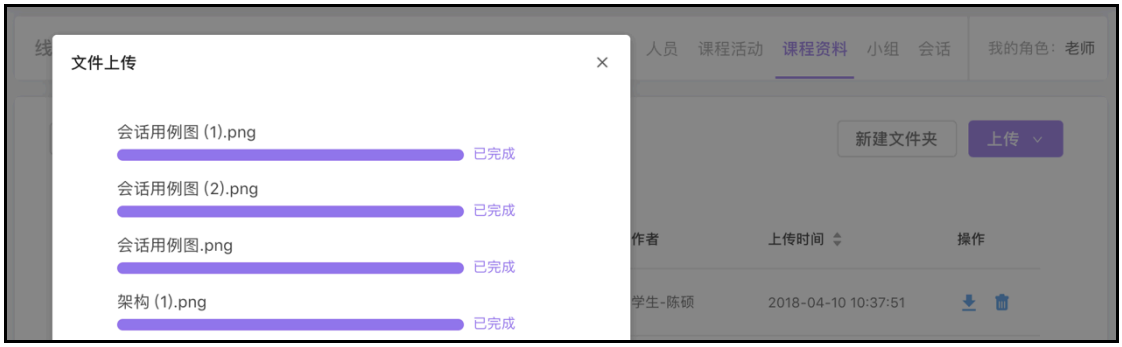

图 5.2 文件批量上传的界面示例

5.2.2 文件夹上传的实现

图 5.3 是文件夹上传的代码实现。上传之前，页面组件首先会对上传的文件数量进行检测，如果超过 MAX_FILE_COUNT 所定义的最大值，会阻止用户上传，提醒用户进行分批上传操作。之后，页面会对上传文件数组的第一个值 inputFiles[0]进行检测，如果 inputFiles[0].webkitRelativePath 字段为 true，代表用户此时进行的是文件夹上传操作。此时，页面会先对文件夹做重名检测，如果所上传的文件夹与列表中已有文件夹重复，会对新文件夹做增量重命名处理。完成之后，页面会首先进行创建新文件夹的操作。页面向服务器提交新建文件夹的请求，服务器会返回新文件夹的 folderId，拿到这个 id 之后，再将用户所选文件夹内的文件上传到该 folderId 的文件夹之下。之后的操作采用递归的方式进行，直到文件夹内所有的文件都上传完成为止。

```
handleUploadFiles = (rawFiles) => {
  if (rawFiles.length > MAX_FILE_COUNT) {
    message.error(`上传文件个数超过${MAX_FILE_COUNT}个，请分开上传！`)
    return
  }
  // 判断是否是文件夹上传
  if (inputFiles[0].webkitRelativePath) {
    // 新建完成后拿到新的 folderId ，将文件夹内的文件上传到该 folderId 下。
    let newFolderName = inputFiles[0].webkitRelativePath.split('/')[0]
    // 判断文件夹重名
    let nth = 0
```

```
    const regex = new RegExp('\^' + newFolderName + '\\(?([0-9]*)\\)?\$')
    fileMap.getIn([path, 'folders']).forEach(v => {
      const regexRes = regex.exec(v.get('name'))
      if (regexRes && +regexRes[1] >= nth) {
        nth = (+regexRes[1]) + 1
      }
    })
    if (nth) {
      newFolderName = newFolderName + "(" + nth + ")"
    }
    this.props.addFolder(affairId, roleId, folderId, newFolderName, path).then(res => {
      let newFolder = _.extend({}, res)
      newFolder['folderId'] = newFolder.id
      delete newFolder['id']
      const newFileMap = this.state.fileMap.updateIn([path, 'folders'], v => v.insert(0,
fromJS(newFolder)))
      this.handleFormatTableData(newFileMap, path)
      // this.setState({ isUploadFolder: true })
      this.handleReadyToUploadFiles(inputFiles, '' + res.id, true)
})}}
```

图 5.3 文件夹上传的代码示例

5.2.3 大文件分片上传的实现

图 5.4 展示了大文件分片上传的代码实现片段。对于视频一类的大型文件，系统前端采用分片上传的方式来提升上传效率。对于 IE10 以上的高级浏览器，HTML5 的 Blob 对象内置了 slice()方法，可以将文件分割成多个部分进行分片操作。分割完成之后，通过 readyToUploadSliceKey 变量存储文件片段的起始坐标 start 和终点坐标 end，通过这两个坐标从文件中取出对应片段进行普通上传操作，并通过_multiProgress 变量存储每个片段的上传结果。如果有某个片段上传失败，页面会终止上传，提醒用户重新上传。页面通过 findNewSliceToUpload() 方法检测各个片段的上传状态，直到所有的片段都上传成功为止。在上传进度渲染方面，页面通过各个片段大小在整个文件中的占比，计算出各部分上传成功之后的进度条推进量，并且每个片段上传时都会实时计算传输量，进度条的渲染是一个平滑的递增过程。

```
uploadSlice(readyToUploadSliceKey, file, uploadId, objectName) {
  let slice = this._multipartUploadVideoProgress.get(readyToUploadSliceKey)
```

```
  const content = file.slice(slice.get('start'), slice.get('end'))
  // 上传碎片
  oss.uploadAffairCover(affair, file, uploadApi, file.name, _objectName, uploadId, readyToUploadSliceKey).then((data) => {
    oss.upload(data.host, this._objectName, uploadId, readyToUploadSliceKey, data.accessId, data.expireTime, data.signature, content).then((etag) => {
      //中断上传
      if(!this._multipartProgress) return;
      // 标记该片上传完成。
      this._multipartProgress = this._multipartProgress.update(readyToUploadSliceKey, v => v.set('isUploading', false).set('etag', etag))
      // 更新进度条。
      this.setState({videoProgress: 100 * (this._multipartProgress.filter(v => v.get('etag') && !v.get('isUploading')).size / this._multipartProgress.size)})
      // 如果所有片都已上传完成，则标志该文件上传完成。
      if (this._multipartProgress.every(v => v.get('etag') && !v.get('isUploading'))) {
        oss.uploadAffairCover(this.props.affair, file, this.state.uploadApi, file.name, "end", this._objectName, uploadId).then((data) => {
          oss.completeMultipartUpload(data.host, objectName, uploadId, data.accessId, data.expireTime, data.signature, this._multipartProgress.map(v => v.get('etag'))).then((res) => {
            return res
          })
        })
        return
      }
      // 开始新的片的上传。
      this.findNewSliceToUpload(file, uploadId, objectName)
    })
  })
}
```

图 5.4 大文件分片上传的代码示例

图 5.5 展示了文件下载的代码片段。由于课程资料通过 OSS 进行存储，且具有一定的下载权限控制，所以下载前需要使用 oss.getDownloadToken()方法获得文件的下载 token。OSS 服务器会返回 token 的下载 url 以及 token 的有效时间。在拿到 token 之后，取出其中的 url，通过 document.createElement()方法在页面创建一个临时的<a>标签，并将<a>标签的 href 属性定义为取回的 url。HTML5 对<a>标签进行了属性扩展，如果标签指定了 download 属性，则标签的点击动作将会触发浏览器的下载行为。所以最终通过 link.click()完成下载，并在下载完成之后销毁临时创建的<a>标签。由于 download 属性只支持部分高级浏

览器，对于版本较低的浏览器，页面采用 location.replace(url)的方法来进行下载。

图 5.5 是大文件分片上传的界面截图。图中展示了视频上传对话框以及浏览器部分网络请求的截图。在进行视频上传时，前端会将视频拆分成多个切片同时进行上传，因此可以看到网络请求中有多个以 mp4 结尾的请求，图中将该视频拆分成了 8 切片，分别对应 8 个网络请求。

图 5.5 大文件分片上传的界面示例

5.2.4 文件下载的实现

图 5.6 展示了文件下载的代码片段。由于课程资料通过 OSS 进行存储，且具有一定的下载权限控制，所以下载前需要使用 oss.getDownloadToken()方法获得文件的下载 token。OSS 服务器会返回 token 的下载 url 以及 token 的有效时间。页面容器在拿到 OSS 返回的文件下载 token 之后，取出其中的 url，通过 document.createElement()方法在页面创建一个临时的<a>标签，并将<a>标签的 href 属性定义为取回的 url。HTML5 对<a>标签进行了属性扩展，如果标签指定了 download 属性，则标签的点击动作将会触发浏览器的下载行为。所以最终通过 link.click()完成下载，并在下载完成之后销毁临时创建的<a>标签。由于 download 属性只支持部分高级浏览器，所以对于版本较低的浏览器，页面采用 location.replace(url)的方法来进行下载。

```
handleDownloadFile = (file) => {
  oss.getDownloadToken(this.props.affairId, this.props.roleId, formData).then(url => {
    // 创建 a 标签
    let link = document.createElement('a')
    if (typeof link.download === 'string') {
      document.body.appendChild(link)
      link.download = ''
      link.href = url
      link.click()
      // 销毁 a 标签
```

```
        document.body.removeChild(link)
      } else {
        location.replace(url)
      }
    })
}
```

图 5.6 文件下载的代码示例

5.3 课程活动模块前端的实现

5.3.1 发布课程活动的实现

图 5.7 是发布课程活动的代码片段，该片段展示了标准的表单提交实现细节。用户在发布课程活动时，页面会渲染活动表单对话框。用户填写完相应字段后，组件首先会调用 form.validateFields()方法给表单内的每个字段做合法性验证。该方法通过回调函数的形式，传回验证结果。如果 err 变量不为空，则代表表单有字段不合法，页面会阻止提交动作，提示用户进行修改；反之，回调函数会进一步对富文本框的文字内容进行“非空”验证。由于富文本输入框支持多个段落的编写，所以在判断过程中，采用 blocks.reduce()方法，将多个段落进行合并之后再做“非空”判断。合法性验证通过之后，表单会将各字段的值赋值给 FormData 对象，且如果活动类型是作业，会额外添加 deadline 和 homeworkType 这两个字段。最终，通过 fetch 对象发送异步请求，完成表单提交动作。

```
handleOk = ()=> {
  // 取出编辑器文本
  const content = this._editor.getWrappedInstance().getContent()
  this.props.form.validateFields(
    (err, values) => {
      // 表单字段合法性判断
      if (!err) {
        // 富文本编辑器内容不为空判断
        const contentText = JSON.parse(content).blocks.reduce((r, v) => r + v.text, '')
        if (contentText.length === 0 || Object.keys(JSON.parse(content).entityMap) === 0) {
          message.error(this.props.title + '内容不能为空')
          return
        }
        const activityType = ACTIVITY_TYPE.indexOf(this.props.title)
        // 活动附件
```

```
        const attachments = values.appendix.map(f => ({fileName: f.name, size: f.size, url: f.response.path}))
        const formObj = {
          content: content,
          type: activityType,
          attachments: attachments,
        }
        if (activityType === 1) {
          formObj['homeworkType'] = values.homeworkType
          formObj['deadline'] = values.deadline.format('x')
        }
        this.setState({ loading: true })
        const { affairId, roleId } = this.props
        this.props.createActivity(affairId, roleId, JSON.stringify(formObj)).then(res => {
          if (res) { this.props.onSuccess() }
          this.setState({ loading: false })
})}}}
```

图 5.7 发布课程活动的代码示例

5.3.2 活动作业提交统计的实现

图 5.8 是活动作业提交统计的代码片段。componentWillMount()方法是课程活动容器的生命周期之一，它在容器被挂载到页面之前执行，其目的是在页面渲染容器之前完成容器数据准备的相关工作。该方法内，会首先获取作业提交列表以及作业整体的提交情况，由于这是两个完全独立的异步请求，请求时长存在先后顺序，所以必须等到最慢的那个请求完成之后，才能进行数据渲染动作，否则页面会出现样式失效等交互性问题。在实现方案上，组件通过 Promise 对象的 all()方法来解决多个请求之间的等待问题。all()方法接受一个请求列表，直到列表内的所有请求都收到服务器响应之后，才会触发 Promise 对象的 then()方法。then()内传入一个请求完成之后的回调函数，如果上述某个请求失败了，页面会在右上角通过 notification 组件弹出相应提示。如果请求全部成功，则可以继续执行容器之后的生命周期，直到完成作业统计的数据渲染。

```
componentWillMount(){
  const {
    affairId, activityId, role, getHomeworkSubmits, getHomeworkStatistics, isGroup
  } = this.props
  // 是否需要统计信息
```

```
  const needStatistics = (role.get('roleType') == USER_ROLE_TYPE.ASSISTANT ||
role.get('roleType') == USER_ROLE_TYPE.TEACHER) && !isGroup
  const getList = getHomeworkSubmits(affairId, role.get('roleId'), activityId)
  // 请求列表
  let promiseList = [getList]
  if (needStatistics) {
    const getStatistics = getHomeworkStatistics(affairId, role.get('roleId'), activityId)
    promiseList.push(getStatistics)
  }
  Promise.all(promiseList).then(res => {
    let homeworkList = this.state.homeworkList
    let statistics = this.state.statistics
    if (res[0].code === 0) {
      homeworkList = fromJS(res[0].data)
    } else {
      notification['error']({
        message: '获取提交作业列表失败',
        description: res[0].data
      })
    }
    if (res[1] && res[1].code === 0) {
      statistics = res[1].data
    } else if (res[1]){
      notification['error']({
        message: '获取作业统计数据失败',
        description: res[1].data
})}}}
```

图 5.8 活动作业提交统计的代码示例

图 5.9 是活动作业提交统计的界面截图。图中上方展示了作业的总参与人数或参与小组数、作业的截止时间、已提交人数、以及未提交人数。下方展示了已提交的作业列表，其中在截止时间之后提交的作业，会打上补交的标签。

图 5.9 活动作业提交统计的界面示例

5.3.3 活动详情版本对比的实现

图 5.10 是活动详情版本对比的代码片段。在对比两段文本之间的差异时，前端使用到了最短编辑距离算法的相关知识[Andoni, 2010]。最短编辑距离算法指出，两段文本之间的差异可以分为新增、替换、以及删除这三种类型。该算法通过动态规划的思路画出文本比对矩阵，进而计算出文本的每一处差异以及差异类型。上述代码片段中，history 对象记录了指定版本的活动详情与当前版本之间的差异，其内部含有 insert、deletion、以及 replacement 三个属性，分别存储了新增内容、删除内容、以及替换内容的比对结果。以新增内容为例，页面对活动详情的每个段落依次进行差异检查，如果段落中存在新增内容，则会在对应内容上做高亮处理。Entity.create()方法会在新增内容上生成高亮组件，且通过 anchorKey、anchorOffset、focusKey、以及 focusOffset 属性来定义高亮内容的光标位置。最终页面会通过这些属性来渲染差异比对结果。类似地，上述代码片段继续检查删除内容和替换内容。

```
export default function differenceHighlighter(contentState, history) {
  if (!history.length) return contentState
  history = history[0]
  if (!history) return contentState
  // 判断新增的内容
  let rawState = convertToRaw(contentState)
  history.insert.forEach((ins) => {
    let newBlocks = rawState.blocks
      .slice(0, ins.position)
      .concat(ins.content)
      .concat(rawState.blocks.slice(ins.position + ins.content.length - 1))
    rawState.blocks = newBlocks
    contentState = convertFromRaw(rawState)
    ins.content.forEach((v) => {
      const entityKey = Entity.create('INSERT', 'IMMUTABLE', {
        text: v.text,
      })
      contentState = Modifier.applyEntity(contentState, new SelectionState({
        anchorKey: v.key,
        anchorOffset: 0,
        focusKey: v.key,
        focusOffset: v.text.length,
      }), entityKey)
```

```
      })
    })
    // 判断删除的内容
    // ...
    // 判断替换的内容
    // ...
}
```

图 5.10 活动详情版本对比的代码示例

图 5.11 是活动详情版本对比的界面截图。活动详情界面的右上方提供查看版本对比按钮。用户点击按钮之后，可以查看当前版本与任意一个较早版本之间差异。界面会在当前的活动详情内容上直接渲染出差异内容，其中黄色代表修改过的内容，绿色代表新增的内容。

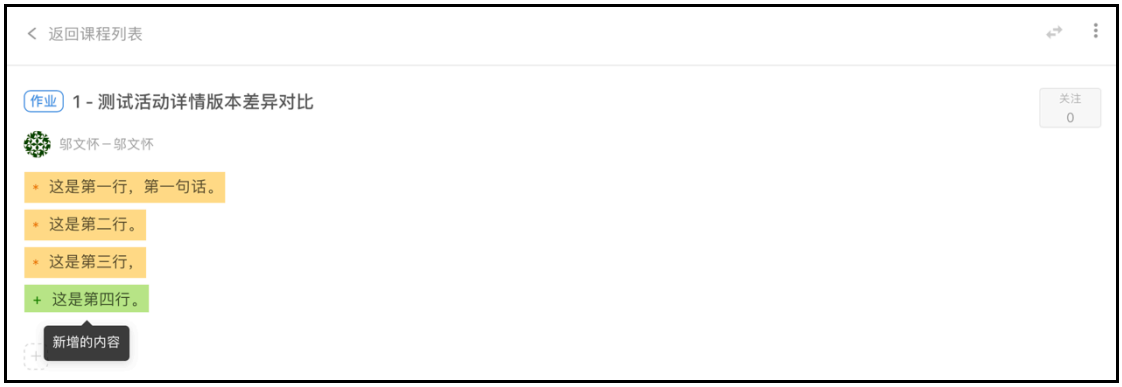

图 5.11 活动详情版本对比的界面示例

5.4 会话模块前端的实现

5.4.1 创建讨论组的实现

图 5.12 是创建讨论组的代码片段。在创建讨论组是，用户需要填写讨论组名称，以及勾选讨论组成员。handleCheckbox()方法负责监听成员列表内每一个 checkbox 的勾选状态，并且将选中列表存储在 checkList 变量内。在成员列表的上方，页面提供搜索输入框，方便用户输入关键字对列表进行过滤。在搜索的过程中，页面将搜索监听函数做了 debounce 防抖动处理。即当用户停止输入的 300ms 之后，页面才会根据用户输入的关键字对列表进行重新渲染，而不是每输入一个字符就过滤一次。这样既提升了页面渲染性能，也完善了用户交互体验。debounce()方法接受两个参数，第一个是需要推迟执行的函数，第二个需要

推迟的时间，它的实现原理是：每次执行函数的请求都会进入等待队列，如果在300ms 的时间间隔内有多次执行函数的请求，则会移除队列内最后一次之前的所有执行请求，并继续开始 300ms 的计时，直到 300ms 后没有新的执行请求，那么就会执行相应函数。在用户勾选完成员之后，由 handleCreate()方法完成提交表单动作。并且在提交之前，会对小组名称进行 trim()处理，即删除字符串两侧多余的空白字符。

```
// 提交创建表单
handleCreate(){
  if (groupName.trim()=== '') {
    this.setState({ groupNameInvalid: 1})
    return
  }
  const groupChat = JSON.stringify({
    containAffairRoles: false,
    name: groupName.trimLeft().trimRight(),
    members: checkedList.map((v) => v.get('id')).toJS()
  })
  this.setState({ loading: true })
  this.props.createGroupChat(groupChat, affairId, roleId).then(res => {
    this.setState({ loading: false })
    if (res) {
      this.props.onCancel()
    }
  })
}
// 勾选人员列表
handleCheckbox(role, event) {
  const { checkedList } = this.state
  const targetIndex = checkedList.findIndex(v => v.get('id') === role.get('id'))
  this.setState({
    checkedList: checked ? checkedList.push(role) : checkedList.splice(targetIndex, 1)
  })
}
// debounce 用户输入，300ms 之后进行角色列表过滤
debounceSetKeyword = _.debounce(keyword => this.setState({ keyword }), 300)
handleTriggerSearchRequest = (e) => {
  this.debounceSetKeyword(e.target.value.trim())
}
```

图 5.12 创建讨论组的代码示例

5.4.2 未读消息提示的实现

图 5.13 是未读消息提示的代码片段。receiveMessageListener()是新消息的监听函数，负责监听 Web Socket 推送过来的最新消息。currentChatKey 变量存储了当前激活的聊天窗口 key 值，监听器内首先会判断新消息是否属于当前激活的聊天窗口。如果属于，则不应做未读提示；如果不属于，将 isUnread 变量置为 1，然后传递给 handleUpdateRecentChat()方法，该方法用来更新会话列表中每一条会话的未读状态。renderChat()方法负责渲染会话列表中的每一项，如果该会话存在未读消息，会在卡片的左上角渲染数字小红点进行提醒，且小红点的渲染不会超过两位数，如果消息未读数超过 99 条，小红点将显示为“99+”。另外，消息卡片上还会渲染最新消息的预览、最新消息的发送时间、以及消息发送对象的头像姓名等用户信息。

```
receiveMessageListener = (newMessage) => {
  const currentChatKey = this.props.selectedChat.get('_key')
  let isUnread = 1
  // 判断是否是当前激活的会话窗口
  if (newMessage.get('_key') === currentChatKey) {
    let { currentMessageList, messageHeights } = this.state
    currentMessageList = currentMessageList.update('msgList', v => v.push(newMessage))
    messageHeights.push(MESSAGE_HEIGHT)
    // 增加面板高度，并滑动至面板最底部
    this.setState({ currentMessageList, messageHeights })
    this.handleScrollToBottom()
    isUnread = 0
  }
  this.handleUpdateRecentChat(newMessage, isUnread)
}
const renderChat = (chat) => {
  // 渲染未读小红点提示
  return (
    <div className={styles.groupChat}>
      <div className={styles.avatar}>
        <img src={group.avatar ? group.avatar : DEFAULT_AVATOR} />
        {unreadCount > 0 ? <span className={classNames('unread', { 'more': unreadCount > 99 })}>{unreadCount > 99 ? '99+' : unreadCount}</span> : null}
      </div>
      <div className={styles.groupInfo}>
```

```
            <div className="name">{group.name}</div>
            <div className="last">{lastMsg ? this.renderLastMessage(lastMsg) : ''}</div>
          </div>
          <div className={styles.time}>
            {lastMsg.time ? relativeTime(lastMsg.time) : ''}
          </div>
        </div>
)}
```

图 5.13 未读消息提示的代码示例

图 5.14 是未读消息提示的界面截图。图中左侧是会话模块的最近会话列表，右侧是聊天面板。用户收到来自其他用户的新消息时，会在最近会话列表中将该消息置顶，并渲染出未读消息数量。另外，“有人@我”等特殊消息也会在消息预览中进行渲染。

图 5.14 未读消息提示的界面示例

5.4.3 消息按需加载的实现

图 5.15 是消息按需加载的代码片段。在消息面板挂载到页面上之后，componentDidMount()方法根据当前面板内可视的消息数量，来定义消息面板的初始高度。如果消息数量较多，面板内会出现滚动条，初始化过程中会自动滚动至最底部。滚动过程由 handleScrollToBottom()方法完成，该方法首先会根据 ref 属性获取面板的 dom 元素，ref 是 React 框架提供的用来方便标记 dom 节点的属性值，获取到相应 dom 元素之后，将该元素的 scrollHeight 属性赋值给 scrollTop 属性，即可完成滑动至最底部的操作。另外，页面用 Inifinite 组件来完成消息面板的渲染工作。当面板内消息数量很多时，页面会维护大量的 dom 元素，但大多数的 dom 元素都是在可见范围之外的，必须通过滑动滚动条才能看见。所以 Infinite 组件根据用户滑动滚动条的行为，来按需加载需要被挂载到页面上的消息 dom 节点。每次滑动至视窗顶部或视窗底部，Infinite 组件会将面板边缘附近的消息预先渲染好，而那些离边缘较远的消息，页面上是没有相应的

dom 节点的。采用这种实现方式大大提升了页面的渲染性能。

```
componentDidMount(){
  // 初始化容器高度
  const containerHeight = this.infiniteContainer ? this.infiniteContainer.offsetHeight : 620
  this.setState({ containerHeight })
  // 滑动至最底部
  this.handleScrollToBottom()
}
// 滚动到底部
handleScrollToBottom = ()=> {
  const inifiniteScrollContainer= findDOMNode(this.scrollContainer)
  if(inifiniteScrollContainer){
    inifiniteScrollContainer.scrollTop = inifiniteScrollContainer.scrollHeight
}}
// 渲染消息面板
renderMessagePanel(){
  return (
    <div className={styles.infiniteContainer} ref={c => this.infiniteContainer = c} >
      <Infinite
        elementHeight={messageHeights}
        containerHeight={containerHeight}
        handleScroll={_.debounce(this.handleScrollTop, 400)}
      >
        { this.renderMessageList()}
      </Infinite>
    </div>
)}
```

图 5.15 消息按需加载的代码示例

5.5 富文本编辑组件的实现

富文本编辑组件在超级账号教学支持系统前端的多个模块中都有使用，是项目中使用频率较高的组件之一。该组件基于 Draft.js 进行编写，Draft.js 提供了丰富的 api 来帮助开发者自主构建 React 风格的富文本编辑器。在数据存储方式上，普通富文本编辑器通常直接存储 html 字符串，而 Draft.js 采用结构化的数据进行存储。该组件将文本内容逐段分开，每段内容需要存储的信息有：该段内容不包含格式的纯文本 plainText、该段的段落样式 blockStyle、该段的行内样式 inlineStyle、以及该段内容所包含的复合实体 Entity（比如超链接所指向的 URL 就是一个 Entity）。

5.5.1 文本段落样式的实现

图 5.16 是文本段落样式的代码片段。blockDataStyleMap 变量定义了段落样式的三种类型：靠左对齐、居中对齐、以及靠右对齐。getBlockStyle()方法会遍历文本的每一个 block，首先通过 block.getData()方法来获取每个 block 的对齐方式，接着通过 block.getType()方法来判断 block 的段落类型：blockquote 代表该段落属于引用段落，unstyled 代表该段落属于普通段落。最后，getBlockStyle()方法会结合以上两个判断来返回每个段落的 CSS 样式类名，从而页面就可以正确渲染出每个段落的样式。

```
// 定义对齐方式 css 类名
const blockDataStyleMap = {
  textAlignment: {
    center: 'alignment-center',
    left: 'alignment-left',
    right: 'alignment-right',
  }
}
export const getBlockStyle = (block) => {
  // 判断段落对齐方式
  const blockDataClassName = block.getData().reduce((reduction, v, k) => {
    if (blockDataStyleMap[k]) {
      reduction.push(blockDataStyleMap[k][v])
    }
    return reduction
  }, [])
  // 判断段落类型
  switch (block.getType()) {
    case 'blockquote':
      return classNames(blockDataClassName, 'RichEditor-blockquote')
    case 'unstyled':
      return classNames(blockDataClassName, 'Main-body')
    default:
      return classNames(blockDataClassName)
  }
}
```

图 5.16 文本段落样式的代码示例

图 5.17 是文本段落样式的界面截图。图中展示了四种段落样式，从上到下分别是靠左对齐、居中对齐、靠右对齐、以及引用段落。前三种段落是普通文本

段落，第四种引用类型在显示上做了特殊处理，段落前通过红色竖线进行区分。

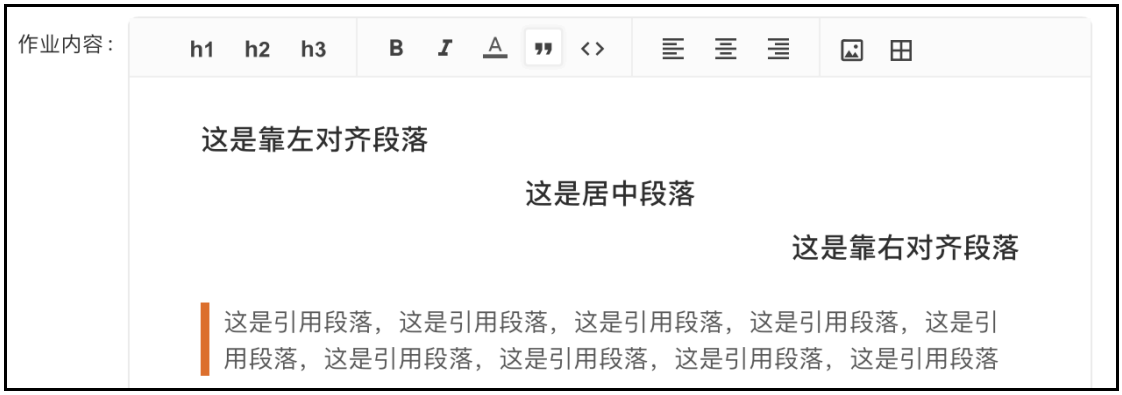

图 5.17 文本段落样式的界面示例

5.5.2 原子段落样式的实现

图 5.18 是原子类型段落样式的代码片段。对于富文本编辑器内插入的图片、视频、表格等特殊类型的内容，Draft.js 也把它当做一个单独的、不可分割的、原子类型的 block 来看待。实现这种特殊类型的 block 渲染，其核心原理是在编辑器区域内渲染出相应类型的组件。getBlockRender()方法会首先获取这个 block 的 entityType 以及 entityData。前者定义了原子段落的具体类型，后者定义类该原子段落所指向的 entity 实体。以图片为例，entityData 指向了该图片的 url，所以最终 getBlockRender()方法会返回相应的图片组件 PhotoComponent。PhotoComponent 组件最终只需要取出 entityData 中的 url 字段，然后根据图片 url 在页面中渲染出对应的<img />标签，即可完成对图片的渲染工作。

```
export const getBlockRender = function(block) {
  // 判断是否是 原子类型 的段落
  if (block.getType()=== 'atomic') {
    const entityType = Entity.get(block.getEntityAt(0)).getType()
    const entityData = Entity.get(block.getEntityAt(0)).getData()
    const editable = this.state.isEditMode !== undefined ? this.state.isEditMode : true
    switch (entityType) {
    // 视频、图片等多媒体类型
    case 'MEDIA':
      if (entityData.type === 'video') {
        return {
          component: VideoComponent,
          editable: false,
        }
```

```
        } else {
            return { component: PhotoComponent, editable: false }
        }
    case 'table':
        // ...
}}}
class PhotoComponent extends React.Component {
  handleImageLoaded = ()=> {
    this.setState({ loading: 'visible' })
  }
  render(){
    const entity = Entity.get(this.props.block.getEntityAt(0))
    // 获取图片地址
    const { src } = entity.getData()
    const { loading } = this.state
    return (
      // 渲染图片
      <div className={styles.imgContainer}>
        <img style={{ visibility: loading }} className={styles.insertPhoto} src={src}
onLoad={this.handleImageLoaded} />
      </div>
)}}
```

图 5.18 原子段落样式的代码示例

图 5.19 是原子段落样式的界面截图。图中展示了两种原子段落，从上到下分别是表格和图片。原子段落在编辑上视作一个整体，只能整段删除或整段插入，即无法只删除表格的某行或某列。

图 5.19 原子段落样式的界面示例

5.5.3 文本行内样式的实现

图 5.20 是文本行内样式的代码片段。行内样式是指与某个段落内的部分内容进行加粗、加下划线、修改字体大小、以及修改字体颜色等操作。以修改字体颜色为例，toggleColor 变量定义了当前编辑器所选择的颜色，selection 变量定义了用户选中的内容范围，这个范围包含了光标的起始位置和终点位置。接着，Modifier 对象通过 removeInlineStyle()方法，将所选范围内的文字的已有颜色样式清除，然后再通过 RichUtils 对象的 toggleInlineStyle()方法，以用户选择的颜色给所选文字着色，并最终返回样式更新后的 editorState。其他类型的行内样式修改原理相同，唯一的区别就是不同的样式具有不同的 CSS style，但是对文字样式的更新过程都是类似的。

```
const toggledColor = this.props.style
// 获取选中内容的坐标范围
const selection = editorState.getSelection()
// 清除该范围内已有的颜色
const nextContentState = Object.keys(COLOR_STYLE_MAP)
  .reduce((contentState, color) => {
    return Modifier.removeInlineStyle(contentState, selection, color)
  }, editorState.getCurrentContent())
let nextEditorState = EditorState.push(
  editorState,
  nextContentState,
  'change-inline-style'
)
const currentStyle = editorState.getCurrentInlineStyle()
if (selection.isCollapsed()) {
  nextEditorState = currentStyle.reduce((state, color) => {
    // 着色
    if (Object.keys(COLOR_STYLE_MAP).indexOf(color)<0){
      if (!state.getCurrentInlineStyle().has(color)){
        state = RichUtils.toggleInlineStyle(state, color)
      }
    } else {
      state = RichUtils.toggleInlineStyle(state, color)
    }
    return state
  }, nextEditorState)
}
```

图 5.20 文本行内样式的代码示例

图 5.21 是文本行内样式的界面截图。图中行内样式主要分为三种，分别是加粗、斜体、以及带有颜色的字体。其中颜色类型的行内样式是互斥的，而其他类型的行内样式是可以相互兼容的，即可以存在加粗并且样式为红色的字体。

图 5.21 文本行内样式的界面示例

5.6 本章小结

本章首先对前端的技术实现做了简要概述。接着通过代码片段以及文字描述的方式，阐述了项目内一些重要功能的技术实现细节。这包括了课程资料模块中文件管理功能的实现细节，课程活动模块中发布课程活动、作业提交统计、以及活动详情版本对比的实现细节，会话模块中创建讨论组、消息提示、以及消息加载的实现细节，还有项目常用组件富文本编辑器在段落样式和行内样式上的实现细节。

第六章 总结与展望

6.1 总结

超级账号教学支持系统以“事务管理模型”为基础。本文设计并实现了超级账号教学支持系统的前端是以 React 为技术核心的单页面 Web 应用，具有功能强大、交互友好、页面性能强等特点。通过超级账号教学支持系统前端，老师可以更方便地组织教学活动，学生可以更有效率地进行课程学习，从而在整体上提高学院和学校的教学质量。

在需求部分，本文首先介绍了系统前端需求的整体概述。接着介绍了课程管理、课程活动、小组管理、课程会话、以及知识体系这五大功能的前端的功能性需求。其中，重点描述了课程管理中创建课程、加入课程、上传课程资料的功能，课程活动中布置作业、生成作业统计的功能，小组 管理中创建小组的功能，课程会话中检索聊天记录、进行视频聊天的功能，以及知识体系构建中编辑知识体系的功能。

在设计部分，本文首先介绍了系统前端的体系架构。之后，本文将系统前端划分为五个模块：课程管理模块、课程活动模块、会话模块、小组模块、以及知识体系模块。然后阐述了每个模块在前端部分的设计思路，给出了各个模块前端设计的时序图或流程图，并对关键类图作出了解释。

在实现部分，本文首先介绍了系统前端的实现概述。接着本文通过代码片段的方式，描述了课程资料模块中文件管理的实现细节，课程活动模块中活动发布、作业提交、活动详情版本对比的实现细节，会话模块创建讨论组、未读消息提示、消息按需加载的实现细节，以及常用组件富文本编辑器的实现细节。

综上所述，超级账号教学支持系统前端以 React 框架为技术核心，界面友好，功能强大，具有很强的页面交互能力和数据展示能力。该系统前端性能优异，能够做到页面按需加载，并通过浏览器缓存策略对部分静态资源做了长期缓存处理，提高了页面加载速度。目前超级账号教学支持系统已经在南京大学软件学院的部分课程中开始试用，从前期的运营数据来看系统运行良好，服务稳定，学校师生在使用过程中也给出了不错的体验反馈。

6.2 进一步工作展望

虽然超级账号教学支持系统前端目前运行良好，但其前端在实际使用的过程中，也存在一些问题有待改进：

首先，富文本编辑组件在进行表格编辑时，无法对表格大小进行动态修改，如果需要增加或删除某行某列，目前只能通过重新插入表格的方式实现。这在用户体验上不够友好，后期的系统前端更新中，将会对表格组件进行优化升级，增加动态修改的操作按钮，争取给用户带来更人性化的操作体验。

其次，目前会话模块中视频聊天的功能，需要 Chrome 浏览器的插件支持才能实现，这在一定程度上增加了用户进行视频聊天的操作成本是使用难度。后期系统前端会寻找相应的替代方案来解决插件依赖的问题，从而降低使用门槛。

参 考 文 献

[程杰, 2007] 程杰，王善利。网络教学支持平台的设计与实现，天津工业大学学报，2007，26(1):61-64。

[陈屹, 2016] 陈屹，深入 React 技术栈，第 1 版，北京：人民邮电出版社，2016。

[陈月, 2016] 陈月，秦福建，Web 前端开发技术以及优化方向探究，信息与电脑(理论版) TP311.52;TP393.09，2016。

[寸志, 2015] 寸志，卓越开发者联盟，React：引领未来的用户界面开发框架，电子工业出版社，2015。

[克里斯, 2017] 克里斯，Web 开发权威指南，第 1 版，北京：人民邮电出版社，2017。

[邓世超, 2017] 邓世超，React 学习手册，第 1 版，北京：中国电力出版社，2017。

[黄勇, 2002] 黄勇，基于 Web 的教学支持系统的开发与应用，山东师范大学，2002。

[李晓峰, 2015] 李晓峰，Web 工程前端性能优化，电子科技，2015，28(5):184。

[祁晖, 2016] 祁晖，底晓强，毕琳。基于 React 的 MOOC 移动学习平台建设研究。教育现代化，2016(38)。

[阮一峰, 2017] 阮一峰，ES6 标准入门，第 3 版，北京：电子工业出版社，2017。

[吴伟敏, 2002] 吴伟敏，网络教学支持平台的研究，硕士论文，南京师范大学新闻传播学院，2002。

[吴浩麟, 2018] 吴浩麟，深入浅出 Webpack，第 1 版，北京：电子工业出版社，2018。

[严新巧, 2017] 严新巧，百俊峰，基于 Dom Diff 算法分析 React 刷新机制，电脑知识与技术，13.18(2017):76-78。

[占东明, 2016] 占东明，Web 新兴前端框架与模式研究，电子商务，

10(2016):65-66。

[张丰麒, 2015] 张丰麒，王飞，ReactJS 的新特性在 Web 开发中的应用，移动信息，2015(10):71-72。

[周俊鹏, 2018] 周俊鹏，前端工程化：体系设计与实践，第 1 版，北京：电子工业出版社，2018。

[周兴宇, 2016] 周兴宇，卞佳丽，基于 React 的前端组件化研究与设计，2016。

[Andoni, 2010] Andoni, Alexandr; Krauthgamer, Robert; Onak, Krzysztof. Polylogarithmic approximation for edit distance and the asymmetric query complexity, IEEE Symp. Foundations of Computer Science (FOCS).

[Berg, 2009] Berg, Alan Mark; Korcuska, Michael. Sakai Courseware Management: The Official guide 1st, Packt Publishing, 2009.

[Bertoli, 2017] Michele Bertoli. React Design Patterns and Best Practices: 1st ed., Packt Publishing 2017.

[Building, 2005] http://www.techsoup.org/, Building Education Websites with Moodle by Joe Rowe, 2000.

[Daniel, 2017] Daniel Bugl. Learning Redux 1st, Packt Publishing, 2017.

[David, 2002] David Gourley, Brian Totty, Marjorie Sayer, Sailu Reddy, Aushu Aggarwal， HTTP Definitive Guide: 1st ed., O'Reilly Media, 2002.

[Flanagan, 2006] Flanagan, David. JavaScript-The Definitive Guide: 5th ed., CA, 2006.

[Freeman, 2011] Adam Freeman. The Definitive Guide to HTML5: 1st ed., Apress 2011.

[Morgan, 2018] Joe Morgan, Simplifying JavaScript: Writing Modern JavaScript with ES5, ES6, and Beyond: 1st ed., Pragmatic Bookshelf, 2018.

[Navarro, 2001] Navarro, Gonzalo. A guided tour to approximate string matching, ACM Computing Surveys. 33(1): 31-88.

[Nicholas, 2017] Nicholas C. Zakas, Understanding ECMAScript 6: 1st ed., No Starch Press, 2016.

[Prusty, 2015] Narayan Prusty. Learning ECMAScript 6: 1st ed., Packt Publishing 2015.

[Silva, 2015] L. H. Silva, M. Ramos, M. T. Valente, A. Bergel and N. Anquetil, "Does JavaScript software embrace classes?," 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Montreal, QC, 2015, pp. 73-82.

[Tilkov, 2015] Tilkov, Stefan,Vinoski, Steve. Node.js: Using JavaScript to Build High-Performance Network Programs, IEEE Internet Computing, 2010, 14(6): 80-83.

[Tuomas, 2017] Juho Tuomas Vepsäläinen, Artem Sapegin, Pedr Browne. SurviveJS - Webpack: From apprentice to master: 1st ed., Packt Publishing, 2017.

致　　谢

首先需要感谢我的指导老师任桐炜，任老师在论文指导过程中要求严格，态度认真，他一丝不苟的学术态度也督促着我在论文书写过程中严格要求自己。

其次我要感谢我在思目的同事管登荣和刘邵。在项目开发过程中，他们给予了我很大的帮助和支持。在论文编写时，他们耐心地为我解答了很多论文书写方面的问题。

另外我要感谢我的舍友，在研究生阶段，大家在学习和生活上一直都互帮互助，和谐友好，共同营造了良好的论文编写环境。

最后再次感谢任老师的辛苦指导，从开题到中期检查，再到最后的论文指导，任老师付出了很多心血，衷心的感谢任老师。

版权及论文原创性说明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名：

日期： 年 月 日