**图片包含 文本

描述已自动生成**

本科毕业设计（论文）

**基于Flutter的跨平台富文本编辑器**

**学 院 软件学院**

**专 业 软件工程**

**学生姓名 孔一涵**

**学生学号 201630106951**

**指导教师 王国华**

**提交日期 2021年 5 月 20 日**

# 摘 要

随着社会生活的信息化和互联网技术的飞速发展，人们对于智能手机的要求越来越高。所以，目前很多APP均需在一定程度上支持文本编辑的功能，而且对此功能的需求不断升高，除了基本的文字输入、预览，还要开始支持文字样式的多样性以及高效排版的功能。

本文利用Flutter技术在Android和iOS双平台上实现了一个结构清晰、API简明、可移植性强、可解析Markdown语法的富文本编辑器软件包。通过该软件包，广大Flutter开发者可以便捷地为自己的APP集成富文本编辑的功能，节省开发时间，提升丰富APP功能。该软件包包括编辑器模块、DOM树模块、Delta模块和Markdown解析模块四大模块。其中编辑器模块是主要和用户交互的模块。用户会通过该模块中的文本框看到自己输入的文本，也可以点击文本框来改变光标位置以及选中区域。同时，用户也要通过该模块中的工具栏进行文字样式的选择。此外，用户通过键盘进行的输入、删除事件也会被编辑器模块监听到并作出及时的处理。而DOM树模块、Delta模块、Markdown解析模块主要为后端的操作，用于对当前文档进行建模、持久化，解析用户的Markdown语法输入，并将最终的结果传递回编辑器模块，使其更新视图。

本文按照软件开发流程对系统的各个模块进行了分析、设计、实现以及测试。目前，本系统可作为一个单独的包集成到其他APP中，使其实现富文本编辑的功能，监听到用户的各种操作，并根据用户操作，来绘制拥有多种样式以及特定排版的富文本。

关键词：富文本编辑器；DOM树；Delta；Flutter

# Abstract

With the informatization of social life and the rapid development of Internet technology, people's requirements for smart phones are getting higher and higher. Therefore, many apps currently need to support the function of text editing to a certain extent, and the demand for this function is increasing. In addition to basic text input and preview, they must also start to support the diversity of text styles and efficient typesetting functions.

This paper uses Flutter technology to implement a rich text editor software package with clear structure, concise API, strong portability, and parsing Markdown syntax on Android and iOS dual platforms. Through this software package, the majority of Flutter developers can easily integrate rich text editing functions for their apps, save development time, and improve rich APP functions. The software package includes four modules: editor module, DOM tree module, Delta module and Markdown parsing module. The editor module is the module that mainly interacts with the user. The user will see the text entered by himself through the text box in the module, or click the text box to change the cursor position and the selected area. At the same time, the user has to select the text style through the toolbar in the module. In addition, the input and deletion events of the user through the keyboard will also be monitored by the editor module and processed in a timely manner. The DOM tree module, Delta module, and Markdown parsing module are mainly back-end operations, which are used to model and persist the current document, analyze user input, and pass the final result back to the editor module to update the view.

This paper analyzes, designs, implements and tests the various modules of the system according to the software development process. At present, this system can be integrated into other APPs as a separate package to realize the function of rich text editing, monitor various operations of users, and draw rich texts with various styles and specific layouts according to user operations.

**Keywords** : Rich text editor；DOM tree；Delta；Flutter

# 目 录

[摘 要 I](#_Toc72782090)

[Abstract II](#_Toc72782091)

[目 录 III](#_Toc72782092)

[第一章 绪论 1](#_Toc72782093)

[1.1 课题背景及研究意义 1](#_Toc72782094)

[1.2 国内外研究现状 1](#_Toc72782095)

[1.2.1 前端富文本编辑器发展研究现状 1](#_Toc72782096)

[1.2.2 Flutter平台富文本编辑器研究现状 2](#_Toc72782097)

[1.3 主要研究工作 3](#_Toc72782098)

[1.4 论文结构 4](#_Toc72782099)

[第二章 基础知识介绍 5](#_Toc72782100)

[2.1 Flutter技术简介 5](#_Toc72782101)

[2.2 Flutter的UI系统 5](#_Toc72782102)

[2.3 Delta 标记语言简介 6](#_Toc72782103)

[2.4 Markdown标记语言简介 7](#_Toc72782104)

[2.5 小结 8](#_Toc72782105)

[第三章 系统功能需求 9](#_Toc72782106)

[3.1 系统的非功能性需求 9](#_Toc72782107)

[3.2 系统的运行环境需求 10](#_Toc72782108)

[3.3 系统的功能性需求 10](#_Toc72782109)

[3.3.1 集成系统用例图 10](#_Toc72782110)

[3.3.2 键盘事件用例分析 11](#_Toc72782111)

[3.3.3 文本框事件用例分析 12](#_Toc72782112)

[3.3.4 工具栏事件用例分析 13](#_Toc72782113)

[3.4 小结 14](#_Toc72782114)

[第四章 系统设计 17](#_Toc72782115)

[4.1 系统结构设计 17](#_Toc72782116)

[4.2 系统流程分析 18](#_Toc72782117)

[4.2.1 键盘事件 18](#_Toc72782118)

[4.2.2 文本框事件 20](#_Toc72782119)

[4.2.3 工具栏事件 20](#_Toc72782120)

[4.3 小结 21](#_Toc72782121)

[第五章 系统实现 23](#_Toc72782122)

[5.1 编辑器部分 23](#_Toc72782123)

[5.1.1 键盘模块 23](#_Toc72782124)

[5.1.2 文本框模块 27](#_Toc72782125)

[5.1.3 工具栏模块 30](#_Toc72782126)

[5.2 DOM树 31](#_Toc72782127)

[5.2.1 DOM树的结构 32](#_Toc72782128)

[5.2.2 DOM树的更新 34](#_Toc72782129)

[5.3 Delta模块 36](#_Toc72782130)

[5.4 Markdown解析模块 36](#_Toc72782131)

[5.5 小结 37](#_Toc72782132)

[第六章 系统测试 37](#_Toc72782133)

[6.1 集成软件包 37](#_Toc72782134)

[6.2 基础页面展示 37](#_Toc72782135)

[6.3 行内样式展示 38](#_Toc72782136)

[6.4 区块样式展示 40](#_Toc72782137)

[6.5 Markdown语法解析功能 40](#_Toc72782138)

[6.6 小结 41](#_Toc72782139)

[结论 43](#_Toc72782140)

[1. 论文工作总结 43](#_Toc72782141)

[2. 工作展望 43](#_Toc72782142)

[参考文献 44](#_Toc72782143)

[致 谢 45](#_Toc72782144)

# 第一章 绪论

## 1.1 课题背景及研究意义

随着社会生活的信息化和互联网技术的飞速发展，智能手机的发展日新月异。以手机APP作为载体完成各类工作的形式也逐渐进入大众视野。同时，在新冠疫情影响背景下，与商务相关的APP下载量更是直线上升，甚至替代了超过了微信、QQ这类常年高居榜首的APP。经过此次疫情，远程工作、在家办公的模式逐渐成熟，可以预见在未来，人们利用移动端处理工作会成为一大趋势。而其中，文本编辑向来是各个行业必不可少的一项基本工作。

同时，除了办公需求，其他主流APP例如，Gmail、QQ邮件等邮件管理APP，知乎、豆瓣、CSDN、虎扑等论坛类APP，微信、微博、Twitter社交类APP，均需在一定程度上支持文本编辑的功能，而且对此功能的需求不断升高，除了基本的文字输入、预览，还要开始支持文字样式的多样性以及高效排版的功能。

Markdown作为一种用于编写结构化文档的纯文本格式，它易读易写的纯文本特性和宽泛简单的语法十分适用于格式化的笔记和说明中，让用户可以不太需要考虑排版问题就写出结构良好的文章。在手机端这种屏幕较小的环境下，利用Markdown语法的进行文本编辑，可以更好更方便的实现排版、存储、分享等功能，提高了编辑效率，改善用户体验。

基于上述讨论，本文旨在利用Flutter技术在android和iOS双平台上提供一个结构清晰、API简明、可移植性强、可解析Markdown文本的富文本编辑器的软件包，提供给广大Flutter开发者，使其可以便捷地为自己的APP集成富文本编辑的功能，节省开发时间，提升丰富APP功能。

## 1.2 国内外研究现状

### 1.2.1 前端富文本编辑器发展研究现状

在前端开发的领域富文本编辑器一直是一大热门，使用非常广泛，很多网页都需要一个内嵌的富文本编辑器。十多年来，世界各地、各类公司推出了多种富文本编辑器的

实现方式，可根据其对浏览器的依赖程度划分为三个阶段。

第一代富文本编辑器代表为CKEditor、UEditor。此代富文本编辑器主要依赖于浏览器原生的编辑能力，用户内容的输入是浏览器直接处理，加粗、斜体、回车等这类的处理则是捕获浏览器的事件来覆盖浏览器默认行为来实现，再辅以一些DOM的嵌套规则和复杂数据输入（如粘贴）的过滤规则来约束数据的正确性。其内容的可编辑主要依赖DOM的contentEditable属性，基于原生execCommand或者自定义扩展的execCommand去操作DOM实现富文内容的修改。

第二代富文本编辑器代表为Quill.js，它的出现给富文本编辑器带了很多新的东西，也是目前开源编辑器里面受众非常大的一款编辑器。此代富文本编辑器底层还是依赖DOM的contentEditable特性，但是对DOM Tree以及数据的修改操作进行了抽象，这意味着编辑器开发者大部分场景下其实不是直接通过修改DOM完成编辑器功能的，而是通过编辑器提供的模型操作API来完成操作的，操作的主角变成了：Delta、Parchment & Blots。此类富文本编辑器对于文本输入的功能还是浏览器的默认行为，编辑器会监控DOM的变化，最终把DOM的更改同步到Delta模型数据中。而对于复杂的样式或者格式操作等非浏览器默认行为，则会直接更新Delta模型数据，由Delta驱动Parchment & Blots的更新，然后最终才到UI的变化。此代富文本编辑器是目前应用最广也最稳定，本文也是基于此代富文本编辑而进行研究设计的。

第三代富文本编辑器目前还没有主流的开源技术代表。此代富文本编辑器可以实现文本的独立排版，不再依靠浏览器的任何编辑功能，自主实现选区光标和内容排版。其实早在2010年Google Doc就使用了这种技术来实现富文本编辑器，但目前还没有一款基于这套架构的开源技术。

### 1.2.2 Flutter平台富文本编辑器研究现状

Flutter自带的TextField输入框，具有多种属性，除了基本的文字样式属性，还包括了一些文本校验、提示文字等属性，满足了大部分APP文本输入的需求。但该输入框在被构造时便限定了文字样式，无法在输入时进行灵活修改，所以整个输入框输入的字体样式统一，无法满足富文本编辑的需求。本文部分实现参考了TextField的结构、功能、方法等。

Zefyr是目前Flutter软件包中较为主流的富文本编辑器包，借鉴了前端比较成熟的富文本编辑器Quill的Delta数据格式，实现了基本的实时编辑功能，包括多级标题、加粗、斜体、引用块、代码块、超链接、插入图片、有序和无序列表等功能。但对于Markdown语法的支持并不完善。目前只支持将输入的文本转化为Markdown、HTML格式进行存储，上传至服务器等，但并不支持Markdown语法的编辑，也不支持Markdown语法的解析，即无法查看.md文件。同时，还缺少一些更高级的功能，例如代码高亮等。而且开发者也放弃维护了，不难想象未来将会有更多的问题出现。

Flutter\_Markdown 为Google官方推出的Markdown解析软件包，可以对Markdown文本进行解析，实现直接在手机上查看Markdown文本的预览。但是功能还不够强大，发展也有些止步不前的趋势，对于代码块的高亮、简单的HTML标签解析都没有做到。而且只是解析器，并不具备编辑功能，只能看，不能修改。

总体来说，在Flutter开发上，无论是对Markdown 的解析、编辑还是富文本编辑器的支持都不够完备，很多功能没有集成，这也是本文研究的意义所在。

## 1.3 主要研究工作

本文主要研究基于Flutter技术的Android和iOS双平台的富文本编辑器插件，用于给Flutter开发者提供一个可集成的、低耦合的软件包。Flutter开发者可以简单、便捷的将本文开发的文本编辑器引入自己的工程中，使其用户可以在APP内进行实时的文本编辑，并自由选择用工具栏或Markdown语法进行进一步排版和设定文字样式。本系统可以分为四个组成模块编辑器模块、Delta模块、DOM树模块，Markdown语法解析模块。

编辑器模块是展示给用户的用户界面，主要由输入文本的文本框和键盘上方的工具栏组成，用户可以方便、简易地选择字体样式，包括行内样式，如选择颜色、选择字体、加粗体、加斜体、加删除线、加下划线的行内样式，以及部分区块样式，如代码块、引用块加特定底板，对于有序和无序列表在换行时自动加序号或中间点“**·**”。对于用户的操作，编辑器会通知给DOM树去更新。

Delta模块负责记录、表达、保存当前的文档样式，主要形式为一组用户的各种动作行为，Delta模块将这些动作行为统一转化为retain、delete、insert三种操作。Delta和DOM树可以相互修改，相互驱动，编辑器直接修改DOM树后，DOM树向当前Detla中添加操作。同时，编辑器也可直接构建一个新的Delta临时模块表明当前的用户操作，用此Delta来驱动DOM树进行更改，再将新的Delta合并到当前Delta中。

DOM树是编辑器绘制页面的依据，是一个并行树结构，有一组子节点，每一个子节点表示用户界面层的一个文字块。当DOM树进行了修改，会通知编辑器进行重新绘制和渲染。

Markdown语法解析模块负责在Markdown语法编辑模式下，用户每次输入完一行文本并输入回车换行后，进行解析。并将结果通知给DOM树，DOM树进行更新，最后反映到编辑器视图层。对于块级的语法，在用户第一次输入回车换行之后新的一行还是保持当前样式，在第二次输入回车时清除自动补全，恢复正常输入状态。

1.4 论文结构

本文一共包括六章，以从理论到实践的思路进行了撰写和分析。在第一章中，首先对课题的背景及研究意义做了简要描述，分析了国内外相关技术的研究现状，并阐述了主要研究工作。在第二章中，对项目的技术背景进行了简单的介绍，首先介绍了Flutter的技术简介及其UI系统，然后介绍了本文涉及到的Delta和Markdown两种标记语言。在第三章中，本文详细阐述了本系统的功能性以及非功能性的需求。在第四章中，根据上一章的需求从系统结构和系统流程上进行了设计。在第五章中，从代码实现的角度，分别对编辑器模块、DOM树模块、Delta模块以及Markdown语法解析模块进行了实现。在最后一章中，以APP开发者的身份通过一个简易Test Demo App展示了本系统的应用效果。

# 第二章 基础知识介绍

本章中将会介绍本系统用到的相关技术的基础知识。本系统利用Flutter技术进行Android 和iOS双端软件包的构建，而在富文本编辑器的设计上运用了Detla标记语言来保存和描述编辑器中的内容数据，同时也支持了Markdown的部分语法的编辑。

## 2.1 Flutter技术简介

目前，占据中国市场的移动端操作系统是Android和iOS。iOS应用采用Objective-C或Swift编写，Android应用采用Java或Kotlin编写。这意味着一个应用想要成为两个移动平台构建和发布应用程序，至少需要两套原生语言。每个平台单独构建和发布应用程序工作量庞大，且费用昂贵[1]。为解决这一系列问题，移动应用跨平台开发各种技术应势而生。续Webview容器和HTML5的Hybrid开发模式，RN、Weex 等方案之后，2018年2月27日Google推出跨平台移动UI框架Flutter，其极强的UI表现力、优秀的跨多端多平台能力、极高的开发与交付效率、良好的开发体验，吸引了众多APP开发团队。目前，原生加Flutter的开发模式也得到了很好的应用，其中包括闲鱼、饿了么、Google ads等。

Flutter的跨平台方案标新立异，既不像Webview容器和HTML5的Hybrid开发模式，利用浏览器技术，又不像RN、Weex使用双端的原生控件，而是依靠高性能渲染引擎Flutter Engine，利用C/C++、Dart和Ski(渲染图形库)构建Widget控件。在安卓平台，Flutter Engine 底层的C/C++代码由NDK编译，而在iOS平台，由LLVM编译。两个平台的Dart代码都是AOT编译为本地代码，Flutter应用程序通过本集指令集运行。Flutter通过相同的渲染器、框架和同一组Widget，达到同时构建Android和iOS应用的目的，无需维护两套独立代码库[2]。

## 2.2 Flutter的UI系统

根据上一小节所述，Flutter构建了多种Widget控件，开发者调用这些控件来进行页面的绘制，但其实最终的Flutter的UI树其实是一个个独立的Element节点构成。而组件最终的布局和渲染都是通过RenderObject来完成的，从开发者创建Widget到真正去渲染的流程为：根据Widget生成Element，然后创建相应的RenderObject并关联到Element.renderObject属性上，最后再通过RenderObject来完成布局排列和绘制。

Element就是Widget在UI树具体位置的一个实例化对象，大多数Element只有唯一的renderObject，但还有一些Element会有多个子节点，如继承自RenderObjectElement的一些类，比如MultiChildRenderObjectElement。最终所有Element的RenderObject构成一棵树，我们称之为‘Render Tree’即‘渲染树’。总结一下，我们可以认为Flutter的UI系统包含三棵树：Widget树、Element树、渲染树。大致关系如图2-1所示。

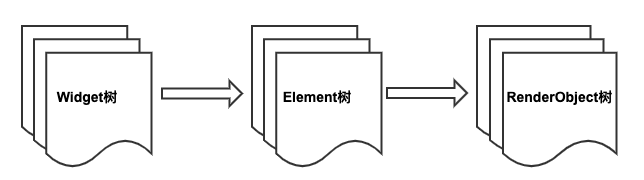


图2-1 Flutter的UI系统

## 2.3 Delta 标记语言简介

Delta是一个扁平的JSON数组，只包含一个 ops 属性，它的值是一个对象数组，每个数组项代表对编辑器的一个操作（以编辑器初始状态为空为基准），以此来保存和描述编辑器中的内容数据，它的变化会直接影响到编辑器中内容的变化。相较于繁重的HTML标记语言，Delta要更加简洁但同时也极富表现力。

Delta只有3种动作和1种属性，却足以描述任何富文本内容和任意内容的变化。其三种操作分别为插入(insert)，删除(delete)，保留(retain)，一种属性为格式属性(attributes)。

如下是一个Delta文件的例子：

{

"ops": [

{

"retain": 12

},

{

"delete": 4

},

{

"attributes": {

"bold": true

},

"insert": "Quill"

},

]

}

其对应的代码为：

const Delta = new Delta().retain(12)

.delete(4)

.insert('White', { "bold": true });

假设在进行该操作之前，编辑器中的内容为“hello world!”。第一个操作retain(12) 表示保留编辑器中索引为0 - 12之间的字符，编辑器中内容不做变动，第二个操作delete(4) 表示接上一个操作后，删除4个字符，编辑器内容变为“hello wo”,第三个操作insert('white', { "bold": true}) 表示接上一次操作后，插入文字“white”, 并且样式为粗体，最终编辑器中的内容为“hello wowhite”。

## 2.4 Markdown标记语言简介

Markdown是一种可以使用普通文本编辑器编写的标记语言，通过简单的标记语法，它可以使普通文本内容具有一定的格式[3]。Markdown的语法简洁明了、学习成本低、排版效果好，而且功能比纯文本丰富。目前，一些主流的大型的在线编辑类的Web平台，例如一些大型博客网站例如CSDN、掘金都可以很好的支持Markdown标记语言。

Markdown语法分为内联元素和块级元素，此处只介绍本系统支持的语法。内联元素包括粗体、斜体、删除线、下划线、分级标题、无序列表、有序列表、引用，对应语法为\*粗体\*、\*\*斜体\*\*、~~删除线~~、++下划线++、# 一级标题（本系统仅支持到三级标题）、\* 无序列表项，- 有序列表项，> 引用。块级元素包括代码块，对应语法为

```

include <stdio.h>

int main(void)

{

printf("Hello world\n");

}

```

鉴于目前markdown标记语言的语法标准不统一，本文使用的标准参考<https://daringfireball.net/projects/markdown/>。

## 2.5 小结

本章首先对Flutter技术进行了介绍，并更深入地介绍了Flutter渲染的过程。然后通过一个简单的例子来详细描述了Delta标记语言，阐明其如何记录一个富文本文档的组成。最后介绍了Markdown标记语言，并明确了本系统所运用到的具体Markdown语法。

# 第三章 系统功能需求

本章先阐述本富文本编辑器软件包非功能性的需求，然后阐明本系统对终端的操作系统以及开发者Flutter版本的需求。最后，通过用例图进行建模，描述了各种用户可能进行的操作，明确系统功能性需求。

## 3.1 系统的非功能性需求

（1）响应性。从用户输入到用户界面的改变，中间经历了控制器修改DOM树，以及用户界面根据新的DOM树进行重新绘制的过程，期间包括很多细粒度的操作。但Flutter 应用程序可编译为原生二进制文件，这些文件依赖于 C/C++ 内置的图形和渲染引擎，因此可生成非常快速且性能出众的应用程序。据统计，Flutter 应用程序能够在大多数设备上实现稳定的 60fps 输出，并在支持高刷新率的设备上实现令人难以置信的 120fps 帧率。在跨平台框架领域，Flutter 在这些指标的竞争中具有明显的优势。

（2）复用性。作为一个开源的软件包，复用性必然是重中之重，本系统将开源到pub.dev上。所有利用Flutter技术进行APP开发的开发者都可以方便简单地将本系统集成到自己的APP中。此外，因为本系统完全开源，APP开发者可以根据自己的需求修改源码，从而增加其他功能、文字样式等。

（3）占用空间小。作为一个待集成系统，占用空间小也是一个重要需求，和集成本系统的APP相比，一定要小很多。为此，对于字体选择功能，本文最终只集成三种字体，防止引入过多的字体库而导致软件包包过大。但APP开发者可以根据需求导入自己的字体库。

（4）稳定性。对于本系统，由于不是APP开发者开发，而是集成到自己的APP中，所以若出了问题，导致整个APP的crash，对其开发者而言将会很难进行debug，必须修改本系统的代码，会耗费大量精力。因此，本系统应该尽可能稳定，做好各种异常情况的处理，防止crash。

（5）易用性。对于本插件，对于使用它的APP开发者来说，应该便于使用，暴露出结构清新明确的构造方法以及参数，方便开发者集成。

## 3.2 系统的运行环境需求

该系统的软件环境要求移动设备的操作系统为Android 系统4.0版本以上，或者iOS10以上，开发环境为Flutter 1.0.0版本以上。

硬件环境要求移动设备具有输入模块、触摸屏、Wi-Fi模块、蓝牙模块等，处理器主频为512MHz以上，内存为256MB以上，闪存为200MB以上。

## 3.3 系统的功能性需求

本小节将通过用例图来描述用户的各种行为，即本系统需要处理的各种用户行为，主要为三大类操作——键盘按动、文本框触碰、工具栏选择。先给出系统的集成用例图，并在接下来的个小节中，对每个事件进行详细的分析说明。

### 3.3.1 集成系统用例图

图3-1是系统的集成用例图，主要描述用户的三类操作，键盘按动、文本框触碰、工具栏选择的用例及其中的关系。

图示

描述已自动生成

图3-1 系统集成用例图

### 3.3.2 键盘事件用例分析

图3-2 展示了用户利用键盘进行各种事件的用例图，具体详见表3-1。

图示

描述已自动生成

图3-2 键盘事件用例图

表3-1 键盘事件用例说明表

|  |  |
| --- | --- |
| 用例名称 | 键盘事件 |
| 用例说明 | 用户在键盘上进行按键操作 |
| 角色 | 用户 |
| 前置条件 | 键盘弹起，用户可以进行输入 |
| 后置条件 | 系统接收到正确的用户输入 |
| 事件流程 | 基本事件流：   1. 用户点击delete键、回车键、其他字符按键 2. 系统接收用户输入的按键 3. 进行相应的页面更新操作   异常事件流：   1. 系统无法检测到用户输入 2. 停在当前状态，系统无反应 |
| 用例场景 | 成功场景：   1. 用户输入后，系统检测到正确的字符进行正确的页面更改   失败场景：   1. 用户等待时间过长，重启APP |

### 3.3.3 文本框事件用例分析

图3-3 为用户对屏幕上展示本文框范围进行各种点击操作用例图，具体详见表3-2。

图示

描述已自动生成

图3-3 文本框事件用例图

表3-2 文本框事件用例说明表

|  |  |
| --- | --- |
| 用例名称 | 文本框事件 |
| 用例说明 | 用户在输入框范围内进行触碰等操作 |
| 角色 | 用户 |
| 前置条件 | 输入框获取当前屏幕焦点 |
| 后置条件 | 系统根据用户操作，进行文本选中或移动光标 |
| 事件流程 | 基本事件流：   1. 用户点击文本框区域内的文字来调整光标位置或者选中一段文字 2. 系统接收到用户的操作 3. 系统根据用户操作来改变文本框界面样式，重新绘制光标位置或者对选中的文字加背景   异常事件流：   1. 系统更改光标或者选中区域位置不正确 |
| 扩展 | 在用户选择部分文字后的操作：   1. 改变此段文字的样式 2. 替换此段文字 3. 删除此段文字 |
| 用例场景 | 成功场景：   1. 用户操作后，系统正确更改文本框界面   失败场景：   1. 用户重新点击文本框区域 2. 系统重新检测 3. 系统修改界面样式 4. 重复1-3直到界面样式正确 5. 若一直不正确，用户重启APP |

### 3.3.4 工具栏事件用例分析

图3-4 展示了用户对屏幕上工具栏的范围进行的点击各种按钮的用例图，具体详见表3-3。

图示

描述已自动生成

图3-4 工具栏事件用例图

表3-3 工具栏事件用例说明表

|  |  |
| --- | --- |
| 用例名称 | 工具栏事件 |
| 用例说明 | 用户点击工具栏的按钮，添加字体样式 |
| 角色 | 用户 |
| 前置条件 | 工具栏展示在当前界面上 |
| 后置条件 | 编辑器添加或取消相应的样式 |
| 事件流程 | 基本事件流：   1. 用户点击工具栏中的按钮 2. 增加或取消对应的样式，进入或退出Markdown编辑模式 3. 相应改变编辑器中的样式 4. 根据编辑器样式修改工具栏按钮状态   异常事件流：   1. 用户点击按钮无反应 |
| 用例场景 | 成功场景：   1. 工具栏样式改变 2. 编辑器中的样式改变，输入下一个字符会应用新的样式或退出/进入Markdown编辑模式   失败场景：   1. 用户重新点击相应的按钮 2. 系统重新检测 3. 系统修改工具栏样式 4. 重复1-3直到界面样式正确 5. 若一直不正确，用户重启APP |

## 3.4 小结

本章先明确了本编辑器的非功能性需求和运行、集成开发的系统环境，然后通过用例图展示了用户可能进行的各种行为，也给出了对应行为编辑器应该进行的操作，为接下来的系统设计明确了目标以及该实现的功能。

# 第四章 系统设计

第三章已对系统的需求进行了详细的分析，明确本系统应完成的任务，本章将根据第三章的系统需求对系统进行设计。首先将对本系统的架构以及各个模块之间的关系进行阐述，然后对各种用户事件，明确处理流程。

## 4.1 系统结构设计

图4-1为系统的整体架构图，其中主要包括四大模块——其中包括：编辑器模块、DOM树模块、Delta模块、Markdown语法解析模块。

图示

描述已自动生成

图4-1 系统整体架构

**编辑器模块**，主要监听用户行为并传递信息。其中视图层包括工具栏、键盘、输入框三个控件，直接展示给用户，控制层为一个控制器，将用户操作传递给其他模块。

**DOM树模块**，是Delta模型和编辑器模块的一个中间层。从Delta记录的用户操作中抽象出了当前文档的样式以及结构并组成一个树结构。当接收到控制器模块或Markdown解析模块的数据，DOM树模块会修改自身结构，并修改Delta模型。当DOM树被更新过后，会通知编辑器根据新的DOM树进行重新绘制。

**Delta模**块，记录通过记录用户的一系列操作来记录文档结构，提供给DOM树模块构建DOM树。同时，编辑模块的控制器也可以直接创建一个Delta类实例，来表示用户的一次操作，由该Delta实例来驱动DOM树修改。

**Markdown语法解析模块**，用于在Markdown编辑模式下，对用户输入进行解析。当用户输入换行符‘\n’，控制器将当前行的文字传递给该模块进行解析，得到用户真实操作，再将结果传递给其他模块。

## 4.2 系统流程分析

通过对上述系统架构的分析，可得知本系统主要处理用户在编辑器视图层进行的键盘事件，包括输入字符、删除字符和换行，文本框事件，包括移动光标和选择字体区域，工具栏事件，包括选择或取消某种样式以及进入或退出Markdown编辑模块。本小节将对上述操作的具体流程进行详细的分析。

### 4.2.1 键盘事件

键盘事件主要包括三种，用户在逻辑键盘上点击了删除键、回车键或其他普通键。

图示

描述已自动生成

图4-2 用户点击删除键流程图

图4-2展示了当用户点击删除键系统的处理流程。首先要判断当前是否有选中区域，若有选中区域则删除选中区域的文字。否则删除当前光标前边的一个字符，若该字符为换行符则返回上一行，否则删除该字符。当删除操作完成之后，要根据当前光标前的文字样式修改编辑器中存放的文字样式使下一个输入和新的最后一个字符的样式相同，若当前光标前无字符，则清空编辑器中的样式。

图4-3展示了当前用户点击回车键的处理流程。当监听到用户点击了回车键输入一个换行符，要先判断前一个字符是否也为换行符，如果是的话，在判断当前是否有块级样式，若有的话，则清楚编辑器中的块级样式。若前一个字符不为换行符，则要判断当前是不是在Markdown编辑模式，如果是，要解析Markdown语法，并根据解析结果重新绘制该行代码。若非Markdown编辑模式，要判断当前编辑器中是否有块级样式，若有的话，则要保持改样式，若没有则要清楚当前编辑器样式。当进行完上述操作后，统一进行换行操作，将光标移动到下一行。

图示

描述已自动生成

图4-3 用户点击回车键流程图

图4-4展示当前用户点击其他按键，输入一个普通字符的处理流程，根据当前编辑器中的样式添加相应的字符。

图示

描述已自动生成

图4-4 用户点击普通字符流程图

### 4.2.2 文本框事件

图4-5展示了用户在屏幕上文本框的范围内进行了操作的流程图。文本框事件主要包括用户点击文本框部分，进行移动光标或选择一块文字区域的操作，而选择一部分文字后可能进行改变样式、替换、删除这三种操作。对于移动光标，只需改变当前编辑器中光标的位置参数，对于选择一块区域，首先要改变当前区域的样式，然后根据用户操作，修改或删除当前文字块。

图示

描述已自动生成

图4-5 用户进行文本框事件流程图

### 4.2.3 工具栏事件

图4-6展示了用户点击工具栏按钮的处理流程。若用户点击了Markdown编辑器按钮，则进入或退出Markdown编辑模式。若用户点击的为字体或颜色选择按钮，则弹出相应的选择框，并根据用户选择更改图标颜色，但图标字体无法更改。若用户点击了块级样式按钮，则其他块级样式按钮置为灰色，表示不可选。在完成上述流程后，工具栏的对应按钮要更改当前的选择态，编辑器中的样式也进行相应的更新。

图示

描述已自动生成

图4-6 用户点击工具栏按钮流程图

## 4.3 小结

本章对系统进行了整体架构设计以及用系统流程分析。在架构设计中，对系统的各模块进行了简单的说明，明确其在整个系统中应负责的部分以及和其他模块之间的关系。在系统流程分析中，对第三章的用例给出了对应编辑器应进行的操作，为接下来的系统实现明确了目标以及不同事件的处理方式。

# 第五章 系统实现

第四章对系统的架构以及主要流程进行了设计，本章将根据第四章的设计对系统的各个模块进行实现。首先，要对编辑器模块进行实现，根据第三章中的各种用户事件，依据第四章中的事件处理流程，实现对应子模块，满足系统功能需求。接着对DOM这个中间层进行实现，然后对Delta模块进行实现，最后对Markdown语法解析模块进行实现。

## 5.1 编辑器部分

编辑器部分是本文中一个难点，本文借鉴了Flutter自带的Textfield输入框，增加了富文本编辑的功能。本编辑根据第三章中的用户事件可以分为几个下属子模块，包括键盘模块、文本框模块、工具栏模块。

### 5.1.1 键盘模块

对于键盘模块主要监听用户的输入，包括输入回车键、删除键以及普通字符，即处理3.3.2的键盘事件。为实现监听键盘的功能，本系统设计了Editor类，使其并实现Flutter官方Service包中的TextInput接口，并重写updateEditingValue方法。TextInput接口将系统键盘和本系统的Editor类实例进行绑定，使系统键盘和Editor类实例之间可以进行通信。当用户通过系统键盘进行输入时，TextInput接口会调用Editor类的updateEditingValue方法，并把新的文本作为参数传递给Editor类实例，Editor类实例再进行进一步的处理。要注意新的文本并非只是用户新输入的文本，而是用户输入完之后，当前全部可编辑的文本，当用户只在旧文本末端进行了一些输入操作，新的文本会包含全部旧的文本。图5-1展示了当用户进行了键盘事件即输入字符等操作的时序图。

图示

描述已自动生成

图5-1 用户进行键盘事件本系统处理方式的时序图

TextInput调用Editor类中updateEditingValue方法的代码如图5-2，其中\_currentConnection.\_client是系统键盘当前绑定的Editor类实例：

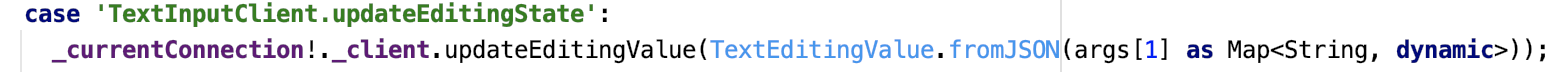


图5-2 TextInput调用Editor中updateEditingValue方法的具体实现代码

图5-3展示了Editor类中updateEditingValue方法的具体实现代码。注意cursorPosition这个参数的赋值，TextEditingValue这个类也是textInput这个接口中实现的，其中可以储存当前文本中一块用户选中的文字区域。如果当前文本有选中区域，则extentOffset偏移量刚好是选中部分的末端，若无选中区域，则extentOffset的偏移量就是当前光标位置。

图形用户界面, 文本, 应用程序

描述已自动生成

图5-3 Editor中updateEditingValue方法的具体实现

图5-3的第五行代码说明Editor类中的updateEditingValue方法会调用controller的getDiff方法，将新的文本和旧的文本进行对比，返回修改开始的位置start，要删掉的文本deleted和新加入的文本inserted。例如，旧文本为“hello connie!”，用户删掉了“connie！”并输入了“world!”，则新文本为“hello world！”。将二者作为参数传递给getDiff方法，可以得到修改开始的位置6，要删掉的文本“connie!”，新加入的文本“world! ”。 接下来，要把这些信息作为参数来调用controller的replaceText方法。图5-4展示了getDiff方法的具体实现代码，第一个for循环用来找到修改结束的位置，第二个for循环用来找到修改开始的位置，如果一样则说明没有要删除掉的文字，只有添加的文字。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图5-4 getDiff方法的具体实现代码

图5-3的最后两行代码说明Editor中的updateEditingValue方法会调用controller的replaceText方法。该方法接受一个开始的位置index，一个要删除的长度len，一个要插入的字符串data，和可能有的被选中区域textSelection作为参数。图5-5为controller中replaceText的具体实现代码。首先要调用document即当前DOM树的replace和compose方法对DOM树进行修改。此外还要判断当前用户输入是否有换行操作，如果有换行操作，要判断当前属性有没有区块级属性，例如codeBlock属性，若有，下一行要继续保持当前属性，若没有，则下一行文字无附加属性。此外，还要根据当前操作是否是对部分文字区域进行修改来调用\_updateSelection方法，改变选中区域。最后所有修改执行完毕调用notifyLinsteners方法，来通知文本框模块根据新的DOM树来更新样式，文本框模块根据DOM树来修改当前界面的步骤将在5.1.2章节中进行详细阐述。至此，用户键盘事件即进行包括换行符在内的各种字符输入删除事件，处理完成，用户可以在屏幕上文本框区域看到修改后的文字了。

图片包含 日程表

描述已自动生成

图5-5 controller中replaceText方法的具体实现代码

### 5.1.2 文本框模块

文本框模块主要解析当前DOM树来展示用户输入的文字，此外，还需要对3.3.3章节中提出的文本框事件进行监听以及处理，主要包括：根据用户在文本框范围内进行点击而修改光标位置以及双击选中部分文字。

对于根据DOM树来展示用户输入的文字这个功能是本文的一个难点。根据2.2章节的介绍，对于Flutter的UI系统，APP开发者大部分情况下只需要使用最顶层Widget，但目前Flutter提供的Widget无法满足本系统的需求，于是本系统需要自己构造RenderObject和其对应的ElementObject来满足更加灵活的渲染需求。基于以上讨论，本系统在该模块中主要实现了EditableTextLine和EditableTextBlock两个控件，即Flutter UI系统中的Widget。

EditableTextLine继承自RenderObjectWidget，表示可以编辑的一行文字。该类为对应的RenderObjectElement 对象提供配置信息，此处EditableTextLine为\_TextLineElement提供配置信息，包括字体、颜色、字体被选择的部分等。EditableTextLine重写了createElement、createRenderObject、updateRenderObject方法，分别用于生成对应的ElementObject对象\_TextLineElement，RenderObject对象RenderEditableTextLine，而RenderEditableTextLine继承自RenderBox，该类详细记录了此RenderObject的物理位置以及其他属性。至此，一个自定义的控件EditableTextLine可以被渲染在终端屏幕上了。

EditableTextBlock继承自MultiChildRenderObjectWidget，表示有多个children的渲染对象，本软件中EditableTextBlock的children 为多个EditableTextLine对象。起到一个组合的作用。图5-6为构造children参数的方法\_buildChildren具体实现代码，利用for循环遍历该block节点中的全部line子节点，构造EditableTextLine。

图片包含 文本

描述已自动生成

图5-6 \_buildChildren方法的具体实现代码

当这两个控件实现之后，本系统就可以根据当前DOM树进行页面的渲染绘制了，执行这一步的主要方法为\_buildChildren，注意这个\_buildChildren不是图5-6中的\_buildChildren，只是功能类似，所以命名相同。在每一次用户利用键盘进行输入，导致DOM树发生改变的时候，都会执行此方法。\_buildChildren接受当前DOM树doc和当前上下文context作为参数，返回一组控件，即Widget。如果DOM树当前节点为Line，则构造一个带有具体样式的EditableTextLine控件，即可编辑的一行文字。如果当前节点为block，则构造一个带具体样式的EditableTextBlock控件，即可编辑的多行相同样式的文字。对于EditableTextBlock还要判断是否为代码块，若是代码块，要在每一行前边留出空间。图5-7展示了\_buildChildren方法主要代码为了表义明确，删除了、合并了部分操作及参数。当得到了这组控件后，Flutter的渲染引擎就可以将其绘制在屏幕上了，用户也能看到自己输入的内容了。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图5-7 根据DOM树修改当前界面的\_buildChildren方法的具体代码实现

对于用户在文本框区域进行的点击事件，修改光标位置以及双击选中部分文字主要通过Delegate类来进行监听。Delegate类引入了Flutter官方手势探测软件包gestures，使其可以监听到用户的各种点击操作。Delegate类的实例会包裹整个文本框区域，当用户进行了点击事件，Delegate类的onTapDown方法将会被调用，将用户具体点击位置的具体信息作为参数传递给文本框的handleTapDown方法，在这个方法中光标的位置将根据用户点击的位置信息来重新计算。图5-8展示了Delegate类的onTapDown方法的具体实现。

图形用户界面, 文本, 应用程序

描述已自动生成

图5-8 onTapDown方法的具体实现代码

而对于用户进行双击选中部分文字的事件，Delegate类的onDoubleTapDown方法将会被调用，将用户双击选中的部分文字作为参数传递给文本框的selectWord方法，在这个方法中将对选中的位置进行重新绘制，在图5-9展示了Delegate类的onDoubleTapDown方法的具体实现。

文本

描述已自动生成

图5-9 onDoubleTapDown方法的具体实现代码

### 5.1.3 工具栏模块

工具栏模块主要由Toolbar类来实现，主要处理3.3.4章节提出的用户工具栏事件，即对屏幕上工具栏中的各种按钮进行点击，以此删除或添加样式，以及进入或者退出Markdown编辑模式。

对于Toolbar类主要实现三种类型的按钮：ToggleStyleButton，SelectHeaderStyleButton，ColorButton。

其中，只有选中态和非选中态的选择按钮ToggleStyleButton涵盖了大部分的样式，例如加粗、下划线按钮，值得注意的是代表是否为编辑模式的Markdown按钮也属于ToggleStyleButton。图5-10展示了ToggleStyleButton的构建方法，其中isToggled参数表示是否被选中，其他参数为按钮本身样式如大小、颜色等。当监听到用户的点击工具栏按钮事件时，isToggled参数取反，并重新绘制按钮样式。

图形用户界面, 文本

描述已自动生成

图5-10 ToggleStyleButton构建方法的具体实现代码

第二种按钮其实将四个ToggleStyleButton组成一组并封装到同一个选择标题按钮SelectHeaderStyleButton中，用于选择当前输入是以标题样式还是普通段落样式。其中四个ToggleStyleButton按钮互斥，即对应的四个isToggled参数每次只有一个为true。当用户每次的选中其中一个按钮，其他三个按钮的isToggled参数也需要进行改变。

第三种按钮则为根据用户操作来修改颜色以及填充颜色的按钮ColorButton。这个按钮比起ToggleStyleButton多了一个fillColor的属性，来表示当前用户选中的颜色，根据这个颜色来更改图标的背景颜色。此外，在点击这个按钮后会跳出颜色选择器，本系统调用了flutter\_colorpicker库。图5-11为ColorButton的构建方法，比起ToggleStyleButton的构建函数，主要多了背景颜色、填充颜色的处理过程。

文本

描述已自动生成

图5-11 ColorButton的构建方法的具体实现代码

同时，所有代表区块级样式的按钮互斥，而行内样式按钮可以叠加，也可以和区块级样式的按钮叠加。

## 5.2 DOM树

DOM树是编辑器模块根据进行界面的渲染绘制的依据，当DOM树发生改变时，编辑器模块的界面要进行相应的更改，同时，在编辑器接收到了用户操作时，也要通知DOM树改变当前的结构，进行新节点的插入，或旧节点的更新、删除。

### 5.2.1 DOM树的结构

DOM树是一个并行的树结构，其中包括root、block、line、leaf四种节点，都继承自抽象类Node，DOM模型基本结构如图5-12。

图示

描述已自动生成

图5-12 DOM树的模型结构

DOM模型涉及到的六个类的具体关系简化类图如下图所示，该类图省略了一些不重要的操作及参数：



图5-13 DOM模型的类图

如图5-13所示，本系统中Node类继承了Dart语言自带LinkedListEntry类，该类是LinkedList类的一个实体。LinkedList类内部的数据结构实现为一个双链表结构，每个节点即一个LinkedListEntry对象，其内部维护了\_next和\_previous指针。

Node类是一个抽象类，Leaf类、Container类是Node类的具体实现。变量parent是其父亲节点，变量style是这个node的样式，变量offset是该节点在其父亲节点的偏移量，变量documentOffset是该节点在整个DOM树中的偏移量，这两个偏移量的单位都为一个字符。toDelta操作是将该节点的文本及样式属性存入Delta模型中。方法Insert, delete, retain是三个抽象方法，由子类进行重写并实现，这三个操作顾名思义对应Delta标记语言的三种操作，在具体实现上，调用了Node重写的insertBefore和insertBefore方法，在当前节点前或后插入一个新的节点。

Leaf类是Node类的一个具体实现，表示文本编辑器中的一个样式统一的文字块，该文字块不可换行，是文本编辑器中渲染的最小单位。比起父类，Leaf类中添加了value这个成员变量，表示该节点存储的字符串。同时，在该类中重写了父类的三个抽象方法insert,delete,retain。

insert方法要先创建一个新的节点，如果要在当前节点的字符串中间插入新自负，则要将该节点分为两个节点，并在两个节点中插入新节点。若要在当前节点的字符串之后插入新字符，则直接在该节点之后插入，无需对当前节点做出更改。插入后再对新节点的样式进行修改。图5-14展示了Leaf类中的insert方法的具体代码实现。

图形用户界面, 文本, 应用程序

描述已自动生成

图5-14 Leaf类中的insert方法的具体代码实现

retain方法主要用于样式的更改以及保留，才会对当前节点进行操作。如果无新样式则直接返回。若有新的样式，则先计算要改变的字符长度和保留不变的字符长度，并把当前节点从开始改变的地方分割，并创造一个新的节点，递归改变新节点的样式，直到没有需要改变的节点为止。图5-15展示了Leaf类中的retain方法的具体代码实现。

图形用户界面, 文本, 应用程序

描述已自动生成

图5-15 Leaf类中的retain方法的具体代码实现

delete方法先计算要删除的字符长度和要删除的字符的位置，用要删除这段字符创建一个新的节点，并从DOM树中删除该节点。Delete具体实现代码如图5-16所示。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成  
图5-16 Leaf类中的delete方法的具体代码实现

Container也是一个抽象类，继承自Node类，因为其本质也是DOM树中的一个节点，但包含了很多叶子结点，对应变量为children。方法childQuery根据参数offset来找到对应的子节点。

Line类是Container类的具体实现，其子节点为Leaf，表示编辑器中的单行文字块。

Block类是Container类的具体实现，其子节点为Line，表示编辑器中的多行文字块。

Root类是DOM树的根节点，其子节点可以为Line、Block。

### 5.2.2 DOM树的更新

根据上一节，编辑器处理用户操作时，主要调用了DOM树的compose方法，以及replace方法。本小结要对这两个方法进行更近一步的解释，说明起内部的实现。

Compose方法主要接受一个新的Delta实例，先要根据这个Delta中的操作来修改DOM树，之后在把这个新的Delta实例中的操作合并到当前表示DOM树的Delta中，图5-17为compose方法的具体实现代码。

**文本

描述已自动生成**

图5-17 DOM类中的compose方法的具体代码实现

replace方法主要接受开始替换的位置，替换掉的长度以及新文字三个参数，主要思路为先构造一个新的包含一个insert操作的Delta实例，并以其作为参数调用compose方法来修改DOM树和当前Delta实例。若替换掉的长度参数大于0，则表示DOM树中有要被删除的部分，此时应构造一个新的包含一个delete操作的Delta实例，以其作为参数调用compose方法来修改DOM树和当前Delta实例。图5-18为replace方法的具体实现。

图形用户界面, 文本, 应用程序

描述已自动生成

图5-18 DOM类中的replace方法的具体代码实现

通过调用这两个方法，编辑器便可以对DOM树进行更改了。在DOM树更改之后，编辑器再根据新的DOM树来重新绘制当前视图。

## 5.3 Delta模块

如上文所述，Delta标记语言本质为一个扁平的JSON数组，只包含一个ops属性，即很多个对文档进行的操作。对于本系统中的Delta类主要功能为记录当前DOM树的构成以及当用户进行操作时，Editor的controller构建一个新的Delta来表示用户操作，并合并到当前记录DOM树的Delta中，从而达到修改DOM树的目的。

Delta类中包含多个Operation类的实例，每一个实例表示对DOM树的一个操作，一共三种类型insert、delete、retain，上文已经介绍过了。

图5-19为Delta类的三种操作retain、inset、delete具体代码实现，retain方法从当前位置 从当前位置保留count个字符，insert方法在当前位置插入数据data，delete方法从当前位置删除count个字符，其中insert和retain可以接受一个Map类型的参数作为样式。

下述push方法要根据当前Delta的最后一个操作，进行处理、合并再更新Delta的操作，例如，若当前Delta中的最后一个操作为insert(‘123’)，新的操作为insert(‘abc’)，则应该将两个操作合并为一个操作insert(‘123abc’)。

*图形用户界面, 文本, 应用程序

描述已自动生成*

图5-19 Delta类中的retain、insert、delete方法的具体代码实现

## 5.4 Markdown解析模块

该模块主要在用户输入回车键之后进行调用，当编辑器检测到了用户输入为‘\n’字符时，首先判断当前样式中，先判断当前是否在Markdown编辑状态，如果是Markdown编辑态，则把当前编辑器中的新文本（具体定义见本章第一节编辑器模块）作为参数，并调用\_parseMarkdown方法进行解析，该方法返回为一个Delta实例。编辑器继续将其作为参数，调用DOM树的compose和replace方法，继续接下来的步骤。

**图形用户界面, 文本, 应用程序

描述已自动生成**

图5-20 Markdown解析模式对“\* ”识别及处理的具体代码实现

以\_parseMarkdown方法中判断是否有行内元素加粗为例，当判断到字符串以'\* '开头， 构建一个Delta先保留之前所有的文本，再删掉当前文本，并插入有加粗样式的，不带‘\*’的文本，最后将光标进行换行。图5-20展示了这部分的代码实现。

当区块级元素被标记为true时，在之后的换行后都会保留当前样式，直到用户连续输入了两次空格。以\_parseMarkdown方法中判断是否有块级元素引用块为例，当判断到字符串以'> '开头，构建一个Delta先保留之前所有的文本，再删掉当前文本，并插入有引用块样式的，不带‘> ’的文本，最后进行换行。图5-21展示了这部分的代码实现。

图形用户界面, 文本, 应用程序

描述已自动生成

图5-21Markdown解析模式对“> ”识别及处理的具体代码实现

## 5.5 小结

本章中根据第四章提出的架构以及系统设计，在本章的前四个小节内分别对编辑器模块、DOM树模块、Delta模块，以及Markdown解析模块四个模块的实现以及和其他模块之间的交互进行了详细的阐述。其中编辑器模块又根据第三章中用户的三种操作用例分出三个子模块来进行相应处理。

# 第六章 系统测试

在本章中，会通过一个简易APP来集成本文实现的软件包，在其页面中创建一个编辑器，展示其界面以及在第三章用例图中展示的各种用户操作对应的表现。

## 6.1 集成软件包

App开发者应先新创建一个新的Flutter工程，并将本系统从github上下载下来，并存放到当前工程的具体路径下，成为一个单独的模块。之后在需要的用该编辑器的文件中进行导入，对应语法为import '$具体路径'。

之后就可以在该文件中创建Editor类、Controller类和Toolbar类的实例，集成一个富文本编辑器了。

## 6.2 基础页面展示

图6-1展示了APP开发者在自己工程的页面中加入本系统，其中Column是一个流式布局的组件，Expanded组件中的组件会自动最大化。

图形用户界面, 文本

中度可信度描述已自动生成

图6-1 APP开发者在自己工程的页面中加入本系统的具体代码

图形用户界面, 表格

中度可信度描述已自动生成

图6-2 基础页面

实现效果如图6-2所示：其中中间大片空白区域为文本框即Editor，键盘上边为工具栏Toolbar。在用户界面设计上风格简洁、表义明确，没有复杂操作，给用户尽可能地提供方便，降低学习使用该功能的成本。

## 6.3 行内样式展示

对于行内样式加黑、斜体、加下划线、加横线、加字体颜色、加背景颜色的实现效果如图6-3a)所示：行内效果也可以叠加使用，集成在一行内。效果如图6-3 b)所示，该图也可以看出某个样式被选中后，工具栏按钮的相应的变化。当用户点击选择字体颜色或填充颜色的按钮后，会弹出颜色选择框，效果如图6-3 c)所示。

当用户点击选择字体的按钮后，弹出字体选择框，为满足尽可能减小空间的非功能性需求，本编辑器只导入了三种字体，Flutter开发者可以根据自己的需求，灵活的导入或删除当前软件包中的字体库。字体选择器实现效果如图6-4 a)所示。



图6-3 a) 行内样式效果 b)行内样式混合使用 c)颜色选择器



图6-4 a) 字体选择器 b) 区块级样式 c) 斜体标题

## 6.4 区块样式展示

对于区块样式多级标题、引用、有序/无序列表、代码块，对于一级标题，根据本章第二小节中的Flutter开发者自定义的样式，所以呈现为蓝色，具体实现效果如图6-4 b)所示。

与行内样式不同，区块级样式不可叠加使用，一段文字只能有一个区块样式属性。但在一个区块文本内，依然可以使用行内元素，例如将标题设置为斜体，实现效果如图6-4 c)所示。然而对于代码块，样式固定为代码形式，不可加行内样式。

6.5 光标移动以及文本区域选择功能展示

对于文本框区域用户可以点击有文字的部分来移动光标，图6-5展示了用户移动光标的操作。

图形用户界面, 应用程序

描述已自动生成

图6-5 a)用户移动光标前 b)用户移动光标后

对于文本框区域用户也可以双击有文字的部分来选定一定的文本区域，图6-6展示了用户选中了部分文本区域的操作。

图片包含 背景图案

描述已自动生成

图6-6 用户选择部分文本区域

## 6.5 Markdown语法解析功能展示

在用户选中了Markdown按钮后，在每一行输入之后，都会对该行文本进行Markdown解析，图6-7为连续测试的几个步骤。



图6-7 a)Markdown效果 b)Markdown效果 c)Markdown效果 d)Markdown效果

## 6.6 小结

在本章中，先从APP开发者的角度，将本文的软件包进行集成，引入到一个简单的APP中。然后以APP使用者的角度来进行第三章用例图中所展示的各种用户操作，并展示了本编辑器对这些操作的处理结果。结果表明，本系统满足了基本富文本编辑的需求，包括根据用户输入和工具栏的样式选择，来展示带有多种样式的、并进行了基本排版的富文本。同时，也支持文本选择以及光标的更改，以及对于基础的Markdown语法也可以进行正确的解析。

# 结论

1. 论文工作总结

从2018年Flutter技术被推出以来，迅速得到了App开发者们的追捧。其独树一帜的跨平台方法，节省了大量的人力成本。更可贵的是，由于其高性能的自研渲染引擎，即使放弃掉原生组件，但性能上的损耗完全在可接受的范围内。所以，在未来，该技术仍然有着广阔的应用场景。但是，由于它出现的时间并不长，很多功能还不具备或并不完善，例如本文实现的富文本编辑功能，仍需要广大App开发者不断进行扩充。

通过对本文的相关技术的研究和系统的实现，得到以下研究成果：

（1）利用Flutter技术，通过一套代码，在iOS和Android两个操作系统上都提供了一个可解析Markdown语法的富文本编辑器系统。实验效果显示该系统具备了基本功能。

（2）本系统利用Flutter Plugin模式开发，使其成为一个插件，其他APP开发人员就可以集成本系统到自己的工程中，实现代码的模块化。但目前部分配置还没成功，审核还未通过，未发布在Pub上。

（3）比起纯iOS和Android开发，Flutter技术仍然很新。该技术领域下，还有大片的空白有待填充。本系统丰富了其对文本编辑的功能，对其底层的代码TextFeild进行了扩充。或许在未来，Flutter的官方技术人员会将本系统的扩充添加到官方代码中。

2. 工作展望

尽管该系统已经满足基本的富文本编辑需求，但在设计和实现过程中仍存在一些不足，在接下来的工作中应将逐步完善。第一，对于将本系统发布到Pub上成为一个正式的插件还需要一定的配置，目前其他APP开发者只能在github上手动下载并添加。第二，在Markdown解析上的方法有些笨拙和单薄，有待改善，后期应封装成单独的模块，进行细化。第三，在富文本支持上，插入图片和视频的功能未能很好的实现，图文混排时的效果和预期不一样，后期将继续改进，找出合适的解决方案。第四，对于撤销功能，没有足够的时间来进行设计及实现，使本系统在使用的上不够方便。在后期将会继续思考其具体实现方案。

# 参考文献

[1] 邓皓瀚.基于Flutter的跨平台移动APP开发前景研究[J].信息与电脑(理论版),2019(15):197-199.

[2] 周勇,程子清. Flutter的原理深度剖析[J]. 电脑编程技巧与维护,2018(11):19-21.

[3] 王伟.标记语言及HTML和XML的比较分析[J].现代图书情报技术,2000,16(5):22-24

[4] 王辰,刘晓鑫,曹晓燕,王佳楠. 基于Vue.js平台的Markdown标记语言插件的研究与实现[J]. 科技风,2018(35):82+85.

[5] 蒋鹏程,李震.基于Flutter框架的快速智能决策app的设计与实现[J].计算机产品与流通,2020(09):107+109.

[6] 宁君宇,袁芳芳,田路强,陈森.基于flutter的移动应用跨平台开发的研究[J].科技风,2020(16):88+94.

[7] 翁子欣,吴明晖.基于Flutter的图片风格转换APP设计与实现[J].计算机时代,2020(02):67-70.

[8]开源富文本编辑器技术的演进（2020 1024）[EB/OL]. <https://zhuanlan.zhihu.com/p/268366406>

[9]正则表达解析Markdown语法 [EB/OL]. http://blog.huanghanlian.com/article/5c80b4176f8b011040530140

[10]用Flutter做个简易的Markdown解析器吧～ [EB/OL]. https://zhuanlan.zhihu.com/p/129632449

[11] 杜文.Flutter实战[M].北京：机械工业出版社，2020:10.

[12] Priyanka Tyagi. Pragmatic Flutter:Building Cross-Platform Mobile APPs for Android, iOS, Web & Desktop[M].CRC Press:2021-03-04.

[13] Frank Zammetti. Practical Flutter[M].Apress, Berkeley, CA:2019-01-01.

[14] Elliott Wen and Gerald Weber. 2018. SwiftLaTeX: Exploring Web-based True WYSIWYG Editing for Digital Publishing. In Proceedings of the ACM Symposium on Document Engineering 2018 (DocEng '18). Association for Computing Machinery, New York, NY, USA, Article 8, 1–10. DOI:https://doi.org/10.1145/3209280.3209522

# 致 谢

本次毕业设计从选题到论文定稿长达近半年，半年里在导师、朋友、家人们的帮助下虽坎坎坷坷但一路成长，最终走到现在，心中自然充满了感激。

感谢我的导师王国华老师，让从未接触过论文和科研的我，从零开始逐渐了解一篇论文撰写的方方面面。在选题初期，我感到迷茫时，王老师利用其丰富的经验和扎实的知识储备为我选定了方向。此后在研究阶段，每次我遇到了困难，王老师都会充分谅解并帮我想出解决办法，和我共同解决问题，给了我坚定的信心和继续钻研下去的勇气。

感谢我在腾讯实习期间的导师以及同事们。在我每次遇到技术上的困难向他们请教时，都会耐心帮助并为我指明方向。同时，他们丰富扎实的专业知识与热情的工作态度让我感受到了这个行业终身学习的活力，为我未来的道路树立了良好的榜样。

感谢这四年来，华工软件学院对我的培养。感谢每一位辛勤的任课老师与辅导员老师们，让这四年收获满满。同时感谢这四年来的同学、朋友，让这四年充满了欢声笑语与难忘回忆，一起相互扶持、相互鼓励的日子会是我人生中熠熠生光的一段回忆。愿大家在未来前程似锦、成为自己想要成为的人。

最后特别地感谢我的父母对我的养育之恩，在漫漫求学路上，是父母一路的支持，让我成为一个独立的人。未来路上，也是我永远的港湾。