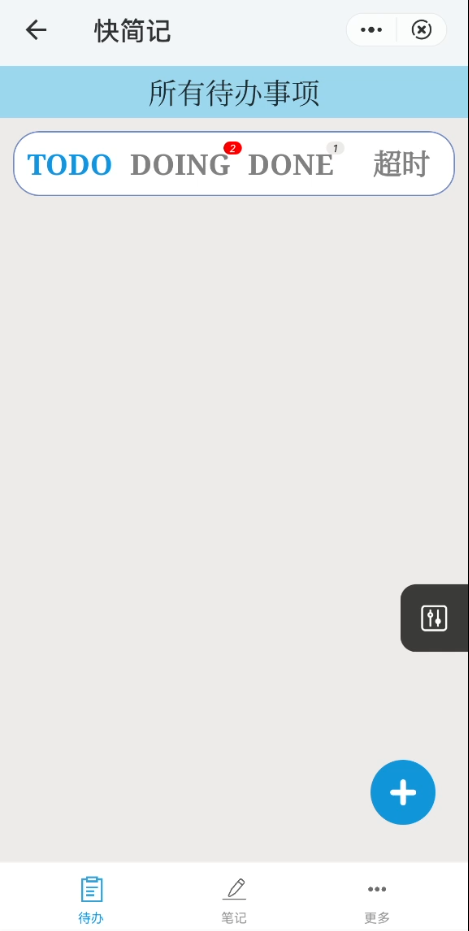
快应用要求和各部分功能实现：

## todo备忘录快应用整体页面的构思与设计



本次实习的任务是做一个todo备忘录，因此在实现相关功能之前，我首先对快应用的页面进行了设计，从需求的角度分析，备忘录除了待办事项之外，最好还要有记笔记的功能，因此我决定将快应用整体分为两个部分，待办事项和笔记，两部分相互独立，分为两个不同的页面，在待办事项和笔记两个页面中实现要求的功能。

整体页面的设计思路如下：

快应用设计的模板使用快应用开发工具提供的备忘录官方模板，页面中包含内容和新建页面，本人在官方模板的基础上添加了需要的功能。

在每一个页面中，我决定将待办事项和笔记以卡片的形式展示，这需要用到快应用官方文档中提供的List功能。结合List和div组件，便可通过屏幕滚动浏览以卡片形式展示的待办事项。 相关的代码如下：

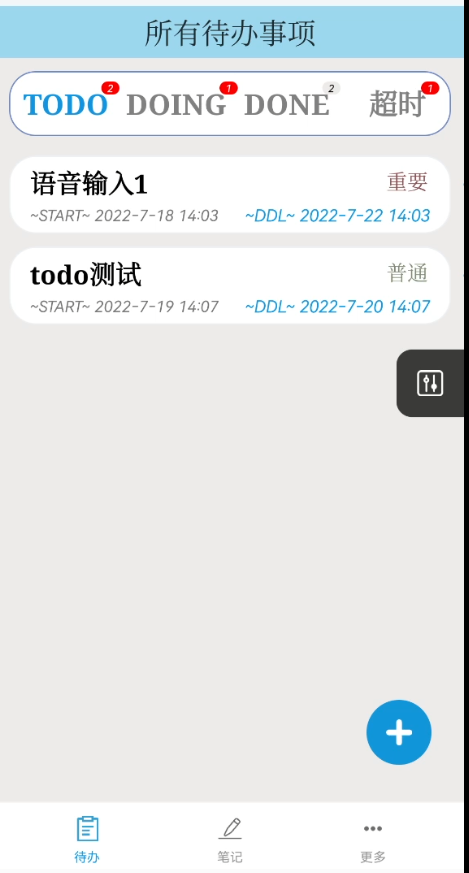


通过使用List，可以将待办事项的卡片以列表的形式展示，而每个具体的卡片则通过子组件实现，其中上图中的<todo></todo>标签便是子组件中形成的卡片的格式，相关代码如下，

根据待办事项的特点，每个待办包括标题、重要程度、开始时间和结束时间四个元素组成，即title、label、start、end。这四个元素由用户在添加页面输入。



具体的效果为：



## 添加页面输入待办事项

根据上文中对待办事项卡片的叙述，用户需要在添加待办的页面输入待办标题、选择重要程度和开始、结束时间。实现方法如下：

标题为纯文字内容，通过快应用官方提供的input标签实现，代码如下



重要程度的标签在三种程度中选择，代码实现如下：

    <!-- 待办事件的重要程度 -->

    <div class="label-wrap">

      <text class="label-title">重要程度:</text>

      <text

        class="{{label === '普通'?'common':''}}"

        onclick="**changeLabel**('普通')"

        >普通</text

      >

      <text

        class="{{label === '一般'?'middle':''}}"

        onclick="**changeLabel**('一般')"

        >一般</text

      >

      <text

        class="{{label === '重要'?'important':''}}"

        onclick="**changeLabel**('重要')"

        >重要</text

      >

    </div>

开始和结束时间则是通过快应用官方提供的时间选择器picker来实现，具体实现如下：

    <!-- 输入开始时间 -->

    <div class='time-area'>

      <text>开始时间:</text>

      <div class='time-input'>

        <text class="data-text">日期:</text>

        <picker class='date-picker' type="date" value="{{start\_date}}" onchange="getStartDate"></picker>

        <text class="data-text">时间:</text>

        <picker class='time-picker' type="time" value="{{start\_time}}" onchange="getStartTime"></picker>

      </div>

    </div>

    <!-- 输入结束时间 -->

    <div class='time-area'>

      <text>结束时间:</text>

      <div class='time-input'>

        <text class="data-text">日期:</text>

        <picker class='date-picker' type="date" value="{{end\_date}}" onchange="getEndDate"></picker>

        <text class="data-text">时间:</text>

        <picker class='time-picker' type="time" value="{{end\_time}}" onchange="getEndTime"></picker>

      </div>

    </div>

添加页面如下：



## 使用kaldi进行语音输入待办事项：

为了实现通过kaldi进行语音输入，首先要在代码文件中导入kaldi语音识别模块，即kaldi语音识别的接口service.asr。并通过模块中的相关函数进行语音识别，将语音识别的结果与待办事项的输入框进行绑定，这样便可实现语音输入，相关代码如下：

import asr from '@service.asr'

**initAsr**(e) {

  let that = this

  asr.**oncompleteresult** = ({ result }) => {

*// this.eventName = this.nameBackUp + `${result}`*

    if(e==1){

        this.title = this.titleBackUp + `${result}`

    }

    if(e==2){

        this.content=this.contentBackUp+`${result}`

    }

  }

},

  startAsr(e) {

    let that = this

*// 通过不同的参数设置语音输入的位置——标题或内容*

    if(e==1){

      this.initAsr(1)

      this.titleBackUp = this.title

    }

    if(e==2){

        this.initAsr(2)

        this.contentBackUp = this.content

    }

*// 调用kaldi语音接口*

    asr.start({

      success: function () {

*//多调用几遍使震动明显一些*

        vibrator.vibrate({ mode: 'short' })

        vibrator.vibrate({ mode: 'short' })

        vibrator.vibrate({ mode: 'short' })

      },

      fail: function (data, code) {

        that.tltle = `start fail, code=${code}, data=${data}`

      }

    })

  },

## 支持修改待办事项

本次开发的待办快应用支持修改待办事项，实现的方法是在用户点击要修改的待办事项时就会进入修改页面，通过将当前待办事项的内容作为参数传进新的页面，用户可以在原来内容的基础上修改，点击保存后会使用新的内容覆盖原来的内容，从而实现待办事项的修改。

相关代码如下：

    <div

*class*="note-item"

*style*="right: {{right}}px;"

*onclick*="**turnToEdit**()"

*ontouchstart*="touchstart"

*ontouchmove*="touchmove"

*ontouchend*="touchend"

*ontouchcancel*="touchcancel"

    >

当用户点击note-item的组件时，也就是待办事项的卡片，就会触发turnToEdit()函数，便可修改待办事项。

turnToEdit()内容如下：

  turnToEdit() {

    router.push({

      uri: 'pages/add/add\_note',

      params: {

        title: this.note.title,

        content: this.note.content,

        index: this.index

      }

    })

  },

通过turnToEdit()函数，将页面切换为新建待办的页面，并将现在该待办事项的内容以参数形式传递进去，例如标题和内容，从而实现待办事项的修改。

## 从手机本地存储、读取数据

因为快应用需要在安卓手机上运行，因此需要将数据存储在手机本地。而快应用官方文档也给出了相关2的接口。通过官方文档的storage模块中的相关函数便可实现数据的存储。

首先需要引入storage模块：import storage from '@system.storage'，通过storage模块中的方法便可实现数据的存储与读取，各种方法及功能如下。

storage.get(OBJECT)——读取存储内容

storage.set(OBJECT)——修改存储内容

storage.clear(OBJECT)——清空存储内容

storage.delete(OBJECT)——删除存储内容

storage.key(OBJECT)——返回存储中某个 index 的键名

为了使代码更加规范，我专门写了一个data.js文件用于数据的存储与读取，代码如下：

import storage from '@system.storage'

export const KEYS = {

  TODO\_LIST\_KEY: 'todo-list-key',

  NOTE\_LIST\_KEY: 'note-list-key',

}

export function **getTodoList**() {

  return new **Promise**((**resolve**, **reject**) => {

**storage**

      .**get**({ key: KEYS.TODO\_LIST\_KEY })

      .**then**(ret => {

        if (ret.data) {

**resolve**(JSON.**parse**(ret.data))

        } else {

**resolve**([])

        }

      })

      .**catch**(err => {

        console.**log**('get TodoList fail', err)

**reject**(err)

      })

  })

}

export function **getNoteList**() {

  return new **Promise**((**resolve**, **reject**) => {

**storage**

      .**get**({ key: KEYS.NOTE\_LIST\_KEY })

      .**then**(ret => {

        if (ret.data) {

**resolve**(JSON.**parse**(ret.data))

        } else {

**resolve**([])

        }

      })

      .**catch**(err => {

        console.**log**('get NoteList fail', err)

**reject**(err)

      })

  })

}

export function **setTodoList**(list) {

**storage**.**set**({

    key: KEYS.TODO\_LIST\_KEY,

    value: JSON.**stringify**(list),

**success**: data => {

      console.**log**('set TodoList success', data)

    },

**fail**: err => {

      console.**log**('set TodoList fail', err)

    }

  })

}

export function **setNoteList**(list) {

*// clearNoteList();*

**storage**.**set**({

    key: KEYS.NOTE\_LIST\_KEY,

    value: JSON.**stringify**(list),

**success**: data => {

      console.**log**('set NoteList success', data)

    },

**fail**: err => {

      console.**log**('set NoteList fail', err)

    }

  })

}

export function **clearTodoList**() {

**storage**.**delete**({

    key: KEYS.TODO\_LIST\_KEY,

**success**: data => {

      console.**log**('delete TodoList success', data)

    },

**fail**: err => {

      console.**log**('delete TodoList fail', err)

    }

  })

}

export function **clearNoteList**() {

**storage**.**delete**({

    key: KEYS.NOTE\_LIST\_KEY,

**success**: data => {

      console.**log**('delete NoteList success', data)

    },

**fail**: err => {

      console.**log**('delete NoteList fail', err)

    }

  })

}

在上述代码中，数据的存储和读取通过6个函数实现，**getTodoList**()、**setTodoList**(list)、**clearTodoList**()三个函数分别用于从手机存储中读取待办事项，将用户输入或修改的待办事项存储到手机以及删除手机存储中的待办事项。

**getNoteList**()、**setNoteList**(list)、**clearNoteList**()，三个函数分别用于从手机存储中读取笔记，将用户输入或修改的笔记存储到手机以及删除手机存储中的笔记。

当需要使用这些存储操作时，只需引用相应的函数而不必在几个代码文件中都写入存储操作，这样可以简化代码，使代码更有逻辑。

## 状态从 todo/done 变成 todo/doing/done

任务要求待办事项的状态可以进行转换，todo状态可理解为未来需要做的待办事项，即事项的开始时间晚于当前时间，同理有还应有doing和done两种状态。但是在后面我考虑到由于自身原因，到了待办事项结束后该待办事项不一定是做完的状态，此时不应该划分为done状态，因此我决定增加一个超时的状态。

因此，对于任意一个待办事项，我将其分为todo、doing、done、超时四类，在每次打开快应用时都会对每个待办事项的开始、结束时间与当前时间进行对比，自动转换待办事项的状态，待办事项的完成需要用户手动勾选。

具体的实现思路是对于一个待办事项，除了上述的标题、重要程度、开始时间、结束时间之外，我为每个待办事项添加了一个新的属性type，type的值为1、2、3、4，分别代表上述的四种状态，具体的实现代码如下：

### 状态的初始化

在每次新建页面时，自动比较用户输入的开始、结束时间与当前时间，确定type的值，代码如下：

 async **addTodo**() {

*// 得到开始时间和结束时间*

    let type=0;

    let start = this.start\_date + ' ' + this.start\_time

    let end =  this.end\_date + ' ' + this.end\_time

*// 根据用户输入的时间对待办事项的状态进行分类*

    let now\_num= (new **Date**()).**getTime**();

    let arr1 = start.**replace**(/[:\-\\&' ']/g, ',').**split**(',');

    let arr2 = end.**replace**(/[:\-\\&' ']/g, ',').**split**(',');

    let start\_num = new **Date**(arr1[0], arr1[1] - 1, arr1[2], arr1[3], arr1[4], 0, 0);

    let end\_num=new **Date**(arr2[0], arr2[1] - 1, arr2[2], arr2[3], arr2[4], 0, 0);

      if(start\_num.**getTime**()>now\_num){

        type=1;*//todo*

      }

      if(now\_num>=start\_num.**getTime**()&&now\_num<=end\_num.**getTime**()){

        type=2;*//doing*

      }

      if(now\_num>end\_num.**getTime**()){*//当前时间已经晚于给出的结束时间，应该重新输入*

          prompt.**showToast**({

              message:'输入时间有误，请重新输入',

              duration:"100000",

*// image:"/Common/logo.png",*

              gravity: 'center'

          })

          router.**push**({

            uri: 'pages/add/add\_todo',

            params: {

                title: this.todo.title,

                index: this.index

              }

          })

      }

其中，可以根据用户的输入为待办事项设置初始状态，其中初始化之后满足条件的待办事项应处于todo或doing状态，其它视为输入错误。即初始化后type为1或2。

### 待办事项状态的更新

    this.**$on**('showPage', this.refreshList)

    const todolist= (await **getTodoList**())

*// 更新事件状态，进行事件统计*

    todolist.**forEach**(function (element, index){

        let arr1 = element.start.**replace**(/[:\-\\&' ']/g, ',').**split**(',')

        let arr2 = element.end.**replace**(/[:\-\\&' ']/g, ',').**split**(',')

        let start\_num =(new **Date**(arr1[0], arr1[1] - 1, arr1[2], arr1[3], arr1[4], 0, 0)).**getTime**();

        let end\_num =(new **Date**(arr2[0], arr2[1] - 1, arr2[2], arr2[3], arr2[4], 0, 0)).**getTime**();

        let now\_num= (new **Date**()).**getTime**();

        let d\_num=end\_num-now\_num;

        if(now\_num>=start\_num&&now\_num<=end\_num && element.type!=3){*//处于doing状态*

            todolist[index].type=2;

            element.type=2;

        }

        if(now\_num>end\_num && element.type!=3){*//超过ddl而且没有完成的话应该是超时状态*

            todolist[index].type=4;

            element.type=4

        }

**todone**(e) {

    this.todoList[e.detail.index].type=3

**setTodoList**(this.todoList);

  },

在上述代码中，使用await方法可以使得todolist的值为**getTodoList**()执行之后的值。

Todolist为从手机存储中读取到的待办事项组成的列表，使用快应用的forEach方法对列表中的每一个事项进行遍历。

如果一个todo事项的开始时间已经早于当前时间，就将其更新为doing事项，type=2；如果一个doing事项的结束时间已经早于当前时间，就将其更新为超时，type=4。

如果一个事项已经被完成，将其更新为done事项，type=3。

如上所述，通过在每次打开快应用时更新每个事项的type属性的值，便可实现待办状态的更新。

## 到 deadline 了进行震动提醒

Todo备忘录需要在每个事项的deadline进行震动提醒。为快应用加上这个功能是我在这次实习中感觉最困难的地方，这个问题困扰了我好久。

完成这一功能需要两步，第一步是调取震动接口，第二步是在恰当的时间调取震动。

首先在快应用中调用震动接口的方法是引入vibrator模块，即：

import vibrator from '@system.vibrator'

通过JavaScript中的setTimeout计时器可以在一段固定时间之后执行函数，而这一段固定的时间在每次启动快应用时计算，求得每个待办距离ddl的时间，当到了ddl之后，就调用三段长震动。代码如下所示。

        if(d\_num>0&&element.type!=3){

*//调试时用到的日志信息*

*// prompt.showToast({*

*//         message:d\_num+"ddl",*

*//         duration:"1000000",*

*//         // image:"/Common/logo.png",*

*//         gravity: 'center'*

*//     })*

*//每次storage.get之后，就设置一遍定时器，进行震动提醒*

**setTimeout**(function(){

*// 调用三段长震动*

              vibrator.**vibrate**({ mode: 'long' })

              vibrator.**vibrate**({ mode: 'long' })

              vibrator.**vibrate**({ mode: 'long' })

*// 返回主页*

              router.**push**({

                  uri: 'pages/main',

              })

*// 弹窗提醒ddl到了*

              prompt.**showToast**({

                    message:"待办事项: "+element.title+" 的ddl到了：",

                    duration:"1000000",

*// image:"/Common/logo.png",*

                    gravity: 'center'

                })

            },d\_num)

        }

## 增加一了个简单统计页面，对待办事项进行统计

收到手机上微信、QQ等软件会在图标右上角的小红点显示未读消息的启发，我打算在菜单栏右上角用小红点表示各类待办事项的个数，实现方法如下。

在对应位置插入数字：

  </div>

    <text *class*="todonum" *if*="{{count(1)!==0}}">{{ count(1) }}</text>

    <text *class*="doingnum" *if*="{{count(2)!==0}}">{{ count(2) }}</text>

    <text *class*="donenum" *if*="{{count(3)!==0}}">{{ count(3) }}</text>

    <text *class*="overtimenum" *if*="{{count(4)!==0}}">{{ count(4) }}</text>

    </stack>

  </div>

数字样式：

/\* 三种事件的数量 \*/

.todonum{

  position: absolute;

  width:14px;

  height:10px;

  top: 58px;

  left: 81px;

  text-align: center;

  font-size: 8px;

  border-radius: 10px;

  color: #ffffff;

  background-color: #ff0000;

  font-family: 'Times New Roman';

  font-style: italic;

}

.doingnum{

  position: absolute;

  width:14px;

  height:10px;

  top: 58px;

  left: 172px;

  text-align: center;

  font-size: 8px;

  border-radius: 10px;

  color: #ffffff;

  background-color: #ff0000;

  font-family: 'Times New Roman';

  font-style: italic;

}

.donenum{

  position: absolute;

  width:14px;

  height:10px;

  top: 58px;

  left: 251px;

  text-align: center;

  font-size: 8px;

  border-radius: 10px;

  color:  #000000;

  background-color: **rgb**(238, 235, 235);

  font-family: 'Times New Roman';

  font-style: italic;

}

.overtimenum{

  position: absolute;

  width:14px;

  height:10px;

  top: 58px;

  left: 327px;

  text-align: center;

  font-size: 8px;

  border-radius: 10px;

  color:  #ffffff;

  background-color: #ff0000;

  font-family: 'Times New Roman';

  font-style: italic;

}

Count函数求出各种待办事项的个数。

**count**(e){

    let num=0;

    const todolist=  **getTodoList**()

    this.todoList.**forEach**(function (element, index){

        if(element.type==e){

          num++;

        }

      }

    )

    return num;

  }

最后效果如下图所示，在菜单栏字符的右上角都会显示数字，表示该类待办事项的数量，还未做完的待办事项会以红点的形式显示。



## 支持一键删除笔记以及待办事项

除了删除指定的待办，在快应用中还提供了一键删除笔记和待办的功能。

代码如下：

  clearTodoList() {

    prompt.showDialog({

      title: '提示',

      message: '是否清空待办',

      buttons: [

        {

          text: '确定',

          color: '#ff2828'

        },

        {

          text: '取消',

          color: '#000000'

        }

      ],

      success: function(data) {

        console.log('handling callback', data)

        if (data.index === 0) {

          clearTodoList()

          prompt.showToast({

            message: '待办已全部清空'

          })

        }

      },

      cancel: function() {

        console.log('handling cancel')

      },

      fail: function(data, code) {

        console.log(`handling fail, code = ${code}`)

      }

    })

  },

清空待办和笔记需要调用上文中提到的clearTodoList()函数和clearNoteList()函数。

## 样式优化、动态效果

在所有功能都实现之后，我进一步优化了整个快应用的样式，整个快应用由底部的菜单栏分成3个独立的界面，在待办事项的界面中又分为四个界面，因此我在待办页的顶部添加了一个导航栏，并且制作了全套快应用的图标，图标整体为蓝灰配色，当选中时为蓝色，未选中是为灰色，整体效果如下所示：



