第 5 章

首 页 开 发

本章将依照绘制好的原型图进行首页开发，在开发首页文字显示功能过程中会介绍如何使用指令在 uni-app 中进行数据绑定，并通过 JavaScript 函数实现首页逐字输出功能及介绍 uni-app 中生命周期的概念。最后在完成首页跳转功能时会为读者介绍 uni-app 中路由跳转的方式。

5.1 使用 HBuilder X 绘制首页

与 Axure RP 绘制流程相同，首先需要确定项目布局及底层页面的样式风格。这里将最外层的 view 标签的 display 属性设置为 flex，将 flex-direction 属性设置为 column，并将其高度定义为 1300rpx。需要注意的是这里的 rpx 长度是小程序特有的长度计量单位，与 px 的换算公式为 1px＝2rpx。最后将这块画布的背景色设置为黑色，代码如下：

```
//第 5 章/index.vue
<template>
    <view class="content">

    </view>
</template>

<script>
    export default {
        data() {
            return {

            }
        },
        onLoad() {

        },
        methods: {

        }
    }
</script>
```

```
<style>
    .content {
        display: flex;
        flex-direction: column;
        background-color: black;
        height: 1300rpx
    }
</style>
```

之后将这个页面用横线一分为二，代码如下：

```
//第 5 章/index.vue
<template>
    <view class="content">
        <view class="middle-line"></view>
    </view>
</template>

<script>
    export default {
        data() {
            return {

            }
        },
        onLoad() {

        },
        methods: {

        }
    }
</script>

<style>
    .content {
        display: flex;
            flex-direction: column;
            background-color: white;
            height: 1300rpx
        }
        .middle-line{
            height: 1rpx;
            width: 1500rpx;
            margin-top: 400rpx;
            margin-left: auto;
            margin-right: auto;
            background-color: black;
        }
</style>
```

再到页面的上半部分进行文字填充,这里可以将原型图的4段文字用一段话描述出来,在每段话之间加入换行符就可以实现段落的效果,代码如下:

```
//第5章/index.vue
<template>
    <view class="content">
        <view>
            <text decode class="info">{{systemInfo}}</text>
        </view>
        <view class="middle-line"></view>
    </view>
</template>

<script>
    export default {
        data() {
            return {
    systemInfo: '欢迎使用Razor-Robot智能工具⚙ \n \n SystemVersion : BetaV1.0.0 \n\n powered
by: g0niw \n \n 请选择功能 :',
        }
    },
        onLoad() {

        },
        methods: {
            }
    }
    </script>

    <style>
        .content {
            display: flex;
            flex-direction: column;
            background-color: white;
            height: 1300rpx
    }
    .middle-line{
        height: 1rpx;
        width: 1500rpx;
        margin-top: 400rpx;
        margin-left: auto;
        margin-right: auto;
        background-color: black;
    }
    .info {
        color: green;
    }
</style>
```

注意这里的systemInfo需要在data选项中定义,其中的\n代表换行符,为了能正常解析出换行符,这里要将text标签添加decode属性以便让\n换行符正常解码,而在这段代码中的emoji字符可以查询https://emojixd.com/网站进行获取。

在编写完上述代码后重新编译，显示的页面如图 5-1 所示。

图 5-1　首页文字显示

接下来再来处理首页的下半部分，和页面的上半部分不同的是首页的下半部分对应了 4 个功能选项，用户在触碰时就要进行对应的功能页跳转，所以需要将这块区域分为 4 个不同的区域，代码如下：

```
//第 5 章/index.vue
<template>
    <view class="content">
        <view>
            <text decode class="info">{{systemInfo}}</text>
        </view>
        <view class="middle-line"></view>
            <text decode style="color: green;">{{choice1}}</text>
            <text decode style="color: green;">{{choice2}}</text>
            <text decode style="color: green;">{{choice3}}</text>
            <text decode style="color: green;">{{choice4}}</text>
    </view>
</template>

<script>
    export default {
        data() {
            return {
                systemInfo: '欢迎使用 Razor-Robot 智能工具 ⚙ \n \n SystemVersion : BetaV1.0.
0 \n\n powered by: g0niw \n \n 请选择功能 :',
                choice1: '一、向我提问 (ChatGPT) \n \n',
                choice2: '二、图片风格化 \n \n',
                choice3: '三、文本翻译 \n \n',
                choice4: '四、实时热点 \n \n',
            }
        },
        onLoad() {

        },
        methods: {

        }
    }
```

```
</script>

<style>
    .content {
        display: flex;
        flex-direction: column;
        background-color: white;
        height: 1300rpx
    }
    .middle-line{
        height: 1rpx;
        width: 1500rpx;
        margin-top: 350rpx;
        margin-left: auto;
        margin-right: auto;
        background-color: black;
    }
    .info {
        color: green;
    }
</style>
```

将上述代码进行编译之后，其呈现出的效果如图 5-2 所示。

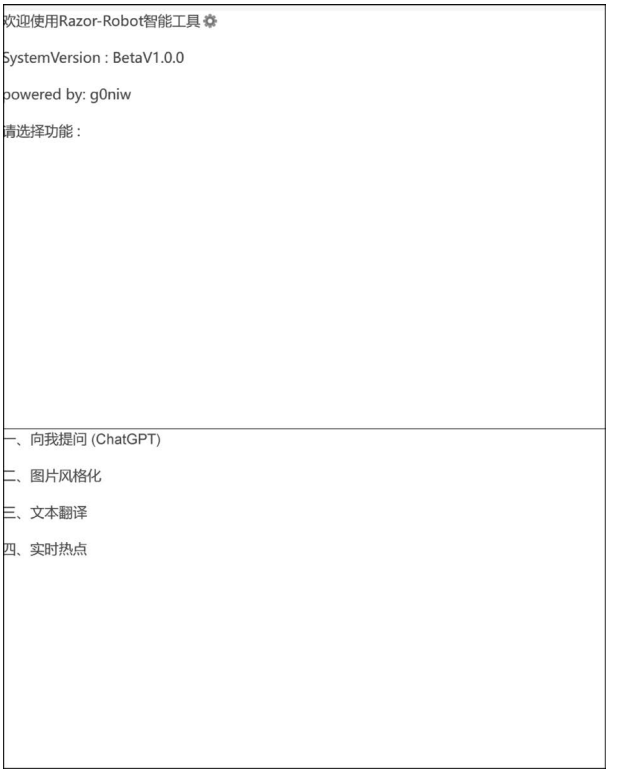

图 5-2　首页下半部分文字显示

首页文字显示的部分已经处理完成了，下面开始填充细节，让这个页面变得可交互。在接下来的章节中会继续案例项目的开发并陆续介绍 uni-app 中绑定数据的指令及它们之前的区别和适用的场景。

5.2　uni-app 中的数据绑定

在元素节点的属性上绑定 data 选项中的数据，不可以直接使用{{ }}插入值语法，此时需要用 uni-app 中的指令来完成操作，其主要用于响应式地更新 HTML 属性，而在 uni-app 中的数据绑定指令主要有 3 种，它们是 v-bind、v-html、v-model。

5.2.1　v-bind 指令

还记得第 4 章中绑定样式的案例吗？现在再来完善这个案例，当前的需求是当单击对应功能选项后出现绿色的选择框，首先修改 text 标签的代码，让其 class 属性动态绑定，代码如下：

```
<text decode style="color: green;" :class="type1"
    @click="choiceT1">{{choice1}}
</text>
```

之后在 data 选项中添加 type1 定义，并在 method 选项中添加 choiceT1 方法，代码如下：

```
//第 5 章/bindType.vue
<script>
    export default {
        data() {
            return {
                type1:''
            }
        },
        onLoad() {

        },
        methods: {
            //当单击触发时该区域样式会被修改为 option
            choiceT1() {
                this.type1 = 'option'
            }
        }
    }
</script>
```

最后在 style 区域内添加 option 对应的样式，代码如下：

```
.option {
    border: 3rpx solid green;
    padding-top: 20rpx;
    margin-top: 10rpx;
    margin-right: 200rpx;
    margin-bottom: 10rpx;
}
```

当单击下方功能栏时该区域样式会被修改,即会出现选择框,其显示效果如图 5-3 所示。

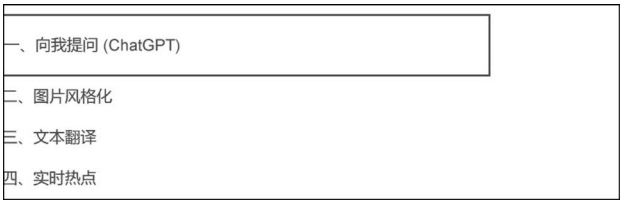

图 5-3 v-bind 绑定效果

5.2.2 v-html 指令

v-html 指令用于将 Vue.js 数据对象中的属性值直接作为 HTML 渲染到模板中,而不是像 v-bind 指令那样简单地绑定属性值。具体来讲,v-html 指令可以在模板中的元素上使用,后面跟随一个表达式,该表达式的值应该是一个包含 HTML 标记的字符串。Vue.js 将会解析这个字符串,并将其作为实际的 HTML 插入模板中,所以此指令不仅可以显示文本内容,还可以显示带标签的内容。例如将实时热点的功能选项以带标签的内容进行绑定,代码如下:

```
//第 5 章/htmlBind.vue
<template>
    <view class="content">
    <view>
        <p v-html="choice4"></p>
    </view>
</template>

<script>
    export default {
        data() {
            return {
                choice4: '<text style="color: green;">四、实时热点</text>',
        }
    },
        onLoad() {

        },
        methods: {

        }
    }
</script>
```

调整过后其显示效果如图 5-4 所示。

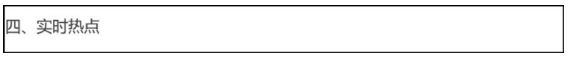

图 5-4 v-html 绑定效果

5.2.3 v-model 指令

v-model 指令主要用于在表单控件元素上创建双向数据绑定。例如用户在登录、注册时

需要提交账号和密码或者用户在检索、创建、更新信息时需要提交一些数据，这些都需要在代码逻辑中获取用户提交的数据，而这些场景通常需要使用 v-model 指令来完成。需要注意的是，v-model 只能用于支持 value 属性和 input 事件的表单元素上，如输入框、复选框和单选按钮。对于其他元素，可以使用 v-bind 实现单向绑定。

这里通过使用输入框并进行回显来讲解该指令的基本用法，代码如下：

```
//第 5 章/vmodel.vue
<template>
    <view>
        <input type="text" v-model="name">
        <text>{{name}}</text>
    </view>
</template>

<script>
    export default {
        data() {
            return {
                name:'test'
            }
        },
        methods: {

        }
}
</script>

<style>

</style>
```

在页面输入框中 name 的值在被修改的同时 data 选项中定义的值也会随之被重新渲染。同样地，修改 data 选项中定义好的 name 的值也会显示在页面上，重新渲染的值如图 5-5 所示。

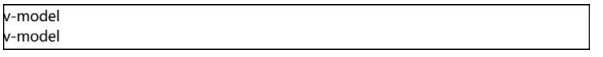

图 5-5　v-model 绑定效果

5.3　在 uni-app 中使用函数

在掌握了 uni-app 中 3 种数据绑定的方式及其适用场景后就可以很从容地完成案例项目中单击出现选择框的功能了。在本节中将继续开发案例项目的首页，完善其功能，并通过 JavaScript 内置函数实现首页文字逐字输出的效果。

5.3.1　函数的定义

在开始编写代码之前，先来了解什么是函数。函数的意思就是由自变量和因变量所确定的一种关系，自变量可能有一个、两个或者 N 个，但因变量的值当自变量确定时也是唯一的。例如 $f(x)=y$ 其中自变量为 x，因变量为 y，在编程领域中，函数是一段可重复使用的代码

块,用于执行特定的任务或操作。函数可以接收输入参数(也称为参数)并返回一个值(也称为返回值),或者仅仅执行一些操作而不返回任何值。而在 uni-app 中可以在 method 选项中进行函数编写,代码如下:

```
//第 5 章/clickFunction.vue
<template>
        <view>
            <text @click = "getType">{{name}}</text>
        </view>
    </template>

    <script>
        export default {
            data() {
                return {
                    name:'单击'
                }
            },
            methods: {
                getType(e){
                console.log(e.type)
                return e.type;
                }
            }
    }
</script>
```

运行上述代码并在页面单击文本框可以在浏览器控制台中看到如图 5-6 所示的打印信息。

图 5-6 自定义函数获取单击事件

可以看到通过定义函数 getType 获取了单击事件的类型名字,其中传参 e 代表 event 事件,而函数返回的 e.type 值则为当前事件的类型。在 method 选项中由开发者定义函数的传参及返回的函数称为自定义函数。还有一类是内置函数,它允许开发者直接调用,以此来完成某些功能而无须关心其逻辑实现。

5.3.2 使用 setInterval 函数实现逐字输出效果

5.3.1 节为读者介绍了函数的基本概念及在 uni-app 中如何编写自定义函数,在本节中会为读者介绍如何通过 JavaScript 内置函数来实现首页页面文字逐字输出的效果。

首先思考这个逐字输出效果应该如何实现,其中有两个关键点:其一是页面上面的文字要以每次追加一个的方式显示出来;其二是每次追加文字时需要有短暂的停顿。关于第一点可以使用分割的方式每次将文字进行分割后通过分割长度累加实现,第二点则可以在字符串长度增加时通过一个定时调用实现,即每次间隔一段时间进行分割并显示,而在 JavaScript 中已经为开发者提供了字符串分割和定时调用的函数,利用这些函数可以实现上述功能,代码如下:

```
//第 5 章/trans.vue
<template>
    <view class="content">
        <view>
            <text decode class="info" selectionchange="true">{{showInfo}}</text>
        </view>
    </view>
</template>

<script>
    export default {
        data() {
            return {
            timer: null,
            showInfo: ''
        }
    },
    onLoad() {
          //声明一个变量,用来监听要分割的长度
          let listenInfoLength = 0
          this.timer = setInterval() => {
          //取到 data.systemInfo 的第 listenInfoLength 位
          this.showInfo = this.systemInfo.substr(0, listenInfoLength);
          //如果 listenInfoLength 大于 data.systemInfo 的长度,则停止计时器
          if (listenInfoLength < this.systemInfo.length) {
                listenInfoLength++
          } else {
                clearInterval(this.timer);
          }
            }, 50)
        },
        methods: {

        }
}
</script>
```

其中,substr 函数可以实现字符串的分割功能。setInterval 函数则可以实现定时调用功能,该函数的第 1 个传参是定时调用的方法,第 2 个传参则是调度时间,其单位为毫秒。还有一点需要注意,该函数是在 onload 选项中编写的,而不是在 method 选项,这么做的原因是因为在 onload 选项中编写的函数在每次进入页面时都会被触发执行,而这些在某些特定的时刻被执行的函数称为生命周期的钩子函数,了解这些钩子函数有利于写出更加简单更加高效的代码。

5.3.3　uni-app 生命周期

在 5.3.2 节中的最后部分为读者介绍了生命周期与钩子函数的概念,在本节中将详细地介绍 uni-app 中的生命周期。与 Vue.js 生命周期类似,uni-app 中的生命周期共有 6 个,它们分别是:应用生命周期、页面生命周期、组件生命周期、模板指令生命周期、Vue 实例生命周期、App 生命周期。这里主要介绍 uni-app 中的 3 个生命周期:应用生命周期、页面生命周期、组件生命周期。

1. 应用生命周期函数

应用生命周期函数只能在 App.vue 文件中监听有效,在其他页监听无效。这些函数包括:①onLaunch,当 uni-app 初始化完成时触发(全局只触发一次);②onShow,当 uni-app 启动或从后台进入前台显示(例如小程序中,用户分享页面再进来就会触发一次 onShow);③onHide,当 uni-app 从前台进入后台时触发;④onError,当 uni-app 报错时触发。

在 App.vue 文件中可以看到这些方法的代码如下:

```
//第 5 章/App.vue
<script>
    export default {
        onLaunch: function() {
            console.log('App Launch')
        },
        onShow: function() {
            console.log('App Show')
        },
        onHide: function() {
            console.log('App Hide')
        }
    }
</script>

<style>
    /* 每个页面的公共 CSS */
</style>
```

当启动并打开应用时在浏览器的控制台中可以看到 AppLaunch 和 AppShow 日志的打印,其具体信息如图 5-7 所示。

图 5-7 程序启动页面展示触发 onLaunch、onShow

此时去浏览别的标签页面,当再次回到该页面时则可以看到如图 5-8 所示的日志打印。

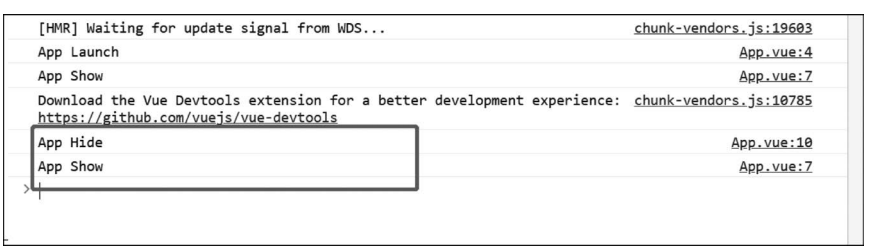

图 5-8 程序隐藏触发 onHide

最后是 onError 函数，首先在 App.vue 文件中定义该函数，代码如下：

```
<script>
    export default {
        onError() {
            console.log("error")
        }
    }
</script>
```

之后编写会引发错误的函数，让函数打印一个未定义的值，代码如下：

```
//第 5 章/error.vue
export default {
    data() {
        return {
            name:'单击'
        }
    },
    methods: {
        add(e){
            console.log(a)
        }
    }
}
```

同样地，当单击页面时会触发函数调用，打开浏览器控制台可以看到如图 5-9 所示的日志输出。

图 5-9　程序运行出错触发 onError

2. 页面生命周期函数

uni-app 页面除支持 Vue.js 组件生命周期外，还支持下面的页面生命周期函数。

1）onlint 函数

该函数用于监听页面初始化，其参数同 onLoad 函数的参数，该参数为上个页面传递的数据，参数类型为 Object(用于页面传参)，该函数的触发时机早于 onLoad。

2）onLoad 函数

onLoad 函数用于监听页面加载，该钩子函数被调用时响应式数据、计算属性、方法、侦听器、props、slots 已设置完成，其参数为上个页面传递的数据，参数类型为 Object(用于页面传参)。该函数的定义及示例代码如下：

```
//第 5 章/onload.vue
<template>
        <view>

        </view>
</template>

<script>
        export default {
            data() {
                return {

                }
            },
            onLoad(e) {
                this.hello()
                console.log(e.data)
            },
            methods: {
                hello(){
                    console.log('hello')
                }
            }
        }
</script>
```

启动应用并在浏览器中访问 http://127.0.0.1:8080/#/pages/index/index?data=123,可以看到在浏览器的控制台中的日志输出如图 5-10 所示。

图 5-10　在 onLoad 中定义函数查看控制台输出

其中,在 onLoad 中编写的 this.hello()每次在加载(刷新)页面时将会自动调用,而其中传递的 e 则可以获取传递给该页面的数据。

3) onShow 函数

该函数会在监听页面时显示(当单击进入其他页面再回来时会触发此函数;如果需要数据变化,则可以使用这个函数),页面每次出现在屏幕上都会触发此函数,包括从下级页面返回当前页面,该函数的示例代码如下:

```
//第 5 章/onShow.vue
<template>
```

```
        <view>

        </view>
</template>

<script>
        export default {
            data() {
                return {

                }
            },
            onLoad(e) {

            },
            onShow() {
                console.log(Date.now())
        }
    }
</script>

<style>

</style>
```

4）onReady 函数

onReady 函数用于监听页面初次渲染完成，此时组件已挂载完成，DOM 树（$el）已可用，注意如果渲染速度快，则会在页面进入动画完成前触发。

5）onHide 函数

该函数可用于监听页面隐藏，该函数可以用于统计用户停留在该页面的时间或者检测用户浏览状态等一些非业务场景的处理。直接使用该函数而不是使用自定义函数将极大地提高程序的性能和代码的健壮性。

6）onUnload 函数

该函数可用于监听页面卸载，例如下述示例，当用户离开页面之后就会获取 getData 的值，其代码如下：

```
    //第 5 章/onLoad.vue
    onLoad(){
        uni.$on("getData",function(e){
console.log(e);             //监听数据
})
    },
    onUnload(){
uni.$off("getData");        //页面卸载时结束监听数据
    }
```

7）onResize 函数

该函数可用于监听窗口尺寸的变化，例如在横屏切换为竖屏时该函数就会被触发。

8) onPullDownRefresh 函数

首先需要在 pages.json 文件中找到对应的 pages 节点,然后在整体的 style 选项中开启 enablePullDownRefresh,将其值设置为 true,如果想让某页不能下拉刷新,则可以在该页的 style 中将 enablePullDownRefresh 设置为 false,而 uni.stopPullDownRefresh 可以停止当前页面的下拉刷新,如果没有使用停止下拉刷新事件,则在页面下拉之后下拉的动画不会自动消失。其示例代码如下:

```
    //第 5 章/onPullDownRefresh.vue
    //首先在 pages.json 文件中开启刷新监听
    {
"pages": [
        {
"path": "pages/index/index",
            "style": {
"navigationBarTitleText": "uni-app",
                "enablePullDownRefresh": true
            }
        }
    ],
    "globalStyle": {
        "navigationBarTextStyle": "white",
        "navigationBarBackgroundColor": "#0faeff",
        "backgroundColor": "#fbf9fe"
}
    }
    //在 pages/index/index.vue 文件中定义方法,在实际开发中延时可根据实际需求来使用
    export default {
        data() {
            return {
                text: 'uni-app'
            }
        },
        onLoad: function (options) {
            setTimeout(function () {
                console.log('start pulldown');
            }, 1000);
            uni.startPullDownRefresh();
        },
        onPullDownRefresh() {
            console.log('refresh');
            setTimeout(function () {
            //1s 后停止页面刷新动画
            uni.stopPullDownRefresh();
        }, 1000);
    }
}
```

9) onReachBottom 函数

该函数用于页面滚动到底部的事件(不是 scroll-view 滚到底),常用于下拉下一页数据。该函数可以在 pages.json 文件中设置具体页面底部触发距离 onReachBottomDistance,如果

由于使用 scroll-view 而导致页面没有滚动，则不会触发触底事件。

注意：在使用 onReachBottom 函数时可在 pages.json 文件里定义具体页面底部的触发距离 onReachBottomDistance，例如设为 50，当滚动页面到距离底部 50px 时就会触发 onReachBottom 事件。

10）onTabItemTap 函数

该函数在单击 Tab 时会被触发，参数为 Object，在使用该函数时具体会返回 3 个属性：①index 属性，该属性类型为 Number 类型，代表被单击 tabItem 的序号，从 0 开始；②pagePath 属性，属性类型为 String，代表被单击 tabItem 的页面路径；③text 属性，属性类型为 String，代表被单击 tabItem 的按钮文字。在 uni-app 中使用该函数的示例代码如下：

```
onTabItemTap : function(e) {
/* e 的返回格式为 JSON 对象: {"index":0,"text":"首页","pagePath":"pages/index/index"} */
    console.log(e);
},
```

注意：onTabItemTap 函数常用于单击当前 tabItem，滚动或刷新当前页面。如果是单击不同的 tabItem，则一定会触发页面切换。如果想在 App 端实现单击某个 tabItem 不跳转页面，则不能使用 onTabItemTap，但可以使用 plus.nativeObj.view 放一个区块盖住原先的 tabItem，并拦截单击事件。

11）onShareAppMessage 函数

该函数在用户单击右上角分享时会被触发，可以用于统计分享信息或者相关分析。

12）onPageScroll 函数

onPageScroll（监听滚动、滚动监听、滚动事件）其参数属性为 scrollTop，属性类型为 Number，该值代表页面在垂直方向已滚动的距离（单位为 px）。需要注意的是在使用 onPageScroll 时不要写交互复杂的 JavaScript，例如频繁修改页面。因为这个生命周期是在渲染层触发的，在非 HTML5 端，JavaScript 是在逻辑层执行的，而两层之间进行通信是有损耗的。如果在滚动过程中频繁地触发两层之间的数据交换，则可能会造成卡顿。

注意：在 App、微信小程序、HTML5 中，可以使用 wxs 监听滚动，而在 app-nvue 中，可以使用 bindingx 监听滚动。

该函数的调用代码如下：

```
onPageScroll : function(e) { //nvue 暂不支持滚动监听,可用 bindingx 代替
    console.log("滚动距离为" + e.scrollTop);
},
```

13）onBackPress 函数

该函数用于监听页面返回，例如返回 event = {from：backbutton、navigateBack}，

backbutton 表示来源是左上角返回按钮或 Android 返回键，而 navigateBack 表示来源是 uni.navigateBack 方法调用。

该函数的使用场景：当页面中的遮罩处于显示状态时，单击返回不希望直接关闭页面，而是隐藏遮罩。遮罩被隐藏后，继续单击返回再执行默认的逻辑。具体的代码如下：

```
    //第 5 章/onBackPress.vue
    //在页面中引入 mask 自定义组件后，通过一种状态值来控制其隐藏/显示
    <mask v-if="showMask"></mask>
    //在 onBackPress 中，判定当前遮罩是否处于显示状态.如果处于显示状态，则关闭遮罩并返回 true
    onBackPress() {
        if(this.showMask) {
this.showMask = false;
return true;
}
    },
```

以上列举了数十种常用的生命周期函数，除了这些函数外 uni-app 还提供了一些用于原生页面的生命周期函数，例如 onNavigationBarButtonTap 函数，用于监听原生标题栏按钮单击事件；onNavigationBarSearchInputChanged 函数，用于监听原生标题栏搜索输入框的输入内容变化事件；onNavigationBarSearchInputConfirmed 函数，用于监听原生标题栏搜索输入框的搜索事件，当用户单击软键盘上的“搜索”按钮时触发。onNavigationBarSearchInputClicked 函数，用于监听原生标题栏搜索输入框的单击事件(只有在 pages.json 文件中的 searchInput 的属性 disabled 被配置为 true 时才会触发)。

3. 组件生命周期函数

uni-app 组件支持的生命周期与 Vue.js 标准组件的生命周期相同。其具体函数名及函数定义见表 5-1。

表 5-1 uni-app 组件生命周期函数名及函数定义

函 数 名	说 明	平台差异
beforeCreate	在实例初始化之前被调用	
created	在实例创建完成后被立即调用	
beforeMount	在挂载开始之前被调用	
mounted	挂载到实例上去之后调用。注意：此处并不能确定子组件被全部挂载，如果需要子组件完全挂载之后执行操作，则可以使用 $nextTickVue	
beforeUpdate	数据更新时调用，发生在虚拟 DOM 打补丁之前	仅 HTML5 平台支持
updated	由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子	仅 HTML5 平台支持
beforeDestroy	实例销毁之前调用。在这一步，实例仍然完全可用	
destroyed	Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁	

在 uni-app 中每个实例在被创建时都要经过一系列的初始化过程，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫作生命周期钩子的函数，这给用户在不同阶段添加自己代码的机会。作为初学者不需要立马弄明白所有的东西，不过随着不断学习和使用，这些钩子函数的参考价值会越来越高。

5.4　uni-app 路由

5.3 节主要介绍了 uni-app 中函数的使用，并着重介绍了 uni-app 中的一类特殊的函数，如生命周期钩子函数，并简要地介绍了这些钩子函数的定义及用法。相信各位读者已经掌握了函数的编写及使用。本节将进行案例项目首页跳转功能的开发并将介绍在 uni-app 中如何使用内置函数（框架封装好的 API）及 navigator 组件进行跳转。

5.4.1　使用 API 进行跳转

uni-app 中的页面路由由框架统一进行管理，开发者需要在 pages.json 文件里配置每个路由页面的路径及页面样式。类似小程序在 app.json 文件中配置页面路由一样，所以 uni-app 的路由用法与 Vue Router 不同，如仍希望采用 Vue Router 方式管理路由，则可以在官方的插件市场 https://ext.dcloud.net.cn 中搜索 vue-router。

1. uni.navigateTo(OBJECT)

使用该方法进行跳转时会保留当前页面，跳转到应用内的某个页面，使用 uni.navigateBack 可以返回原页面。该方法的参数说明见表 5-2。

表 5-2　uni.navigateTo 参数说明

参　数	类型	是否必填	说　明
url	String	是	需要跳转的应用内非 tabBar 的页面的路径，路径后可以带参数。参数与路径之间使用“?”分隔，参数键与参数值用“=”相连，不同参数用“&”分隔，例如 'path? key=value&key2=value2'
animationType	String	否	窗口显示的动画效果
animationDuration	Number	否	窗口动画持续时间，单位为 ms
events	Object	否	页面间通信接口，用于监听被打开页面发送到当前页面的数据
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

回到案例项目首页，在单击下方功能选项时需要跳转到对应的页面。首先来创建功能页面，选择 index 文件夹右击，选择“新建页面”后会弹出如图 5-11 所示的新建页面。

创建完成之后该页面组件会自动地在 page.json 文件中进行注册。之后会看到项目中多出了一个名为 chat 的页面，如果勾选了自动注册，则会在 page.json 文件中完成自动配置，新建完成之后项目的目录结构如图 5-12 所示。

图 5-11　新建 uni-app 页面

图 5-12　chat 页面注册

在完成了上述操作之后就可以编写路由跳转的代码了，这里以 uni.navigateBack 方法为例，完成 index 页面跳转到 chat 页面的功能，代码如下：

```
//第 5 章/index.vue
//添加单击事件
<text decode style = "color: green;" :class = "type1"
@click = "choiceT1">{{choice1}}
</text>
//在单击事件中添加跳转逻辑
methods: {
    choiceT1() {
        this.type1 = 'option'
        uni.navigateTo({
            url: '/pages/index/chat',
            success(res) {
            console.log("成功跳转 chat 页面")
            }
        })
    }
}
```

在单击功能项后可以在浏览器控制台中看到其日志输出，如图 5-13 所示。

图 5-13　chat 页面跳转成功日志

这里需要注意的是传参 url 有长度限制，太长的字符串会传递失败，可改用窗体通信、全局变量，另外当参数中出现空格等特殊字符时需要对参数进行编码，可使用 encodeURIComponent 方法对参数进行编码，代码如下：

```
<navigator :url = "'/pages/test/test?item = ' +
encodeURIComponent(JSON.stringify(item))"></navigator>
```

而在传参接收页面（跳转目的页面）应该使用 decodeURIComponent 函数来接收传参，代码如下：

```
//在跳转页面接收传参
onLoad: function (option) {
    const item = JSON.parse(decodeURIComponent(option.item));
}
```

注意：使用该方法由于会将页面存入页面栈，所以其页面跳转路径有层级限制，不能无限制地跳转新页面（不断跳转新页面会导致页面栈占满，从而导致程序异常），而且路由 API 的目标页面必须是在 pages.json 里注册的页面。

2. uni.redirectTo(OBJECT)

使用该方法进行跳转会关闭当前页面,跳转到应用内的某个页面。该方法的参数说明见表 5-3。

表 5-3 uni.redirectTo 参数说明

参 数	类 型	是否必填	说 明
url	String	是	需要跳转的应用内非 tabBar 的页面的路径,路径后可以带参数。参数与路径之间使用"?"分隔,参数键与参数值用"="相连,不同参数用"&"分隔,例如'path? key=value&key2=value2'
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数(调用成功、失败都会执行)

修改上个案例中的代码,使用 uni.redirectTo 方法同样能够完成页面跳转,示例代码如下:

```
/第 5 章/ redirectTo.vue
methods: {
    choiceT1() {
        this.type1 = 'option'
        uni.redirectTo({
        url: '/pages/index/chat',
        success(res) {
            console.log("成功跳转 chat 页面")
          }
        })
    }
}
```

注意:跳转到 tabBar 页面只能使用 switchTab 跳转。

3. uni.reLaunch(OBJECT)

使用该方法进行跳转会关闭所有页面,然后打开应用内的某个页面。该方法的参数说明见表 5-4。

表 5-4 uni.reLaunch 参数说明

参 数	类 型	是否必填	说 明
url	String	是	需要跳转的应用内非 tabBar 的页面的路径,路径后可以带参数。参数与路径之间使用"?"分隔,参数键与参数值用"="相连,不同参数用"&"分隔,例如'path? key=value&key2=value2'
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数(调用成功、失败都会执行)

使用 uni.reLaunch 方法同样能够完成页面跳转,示例代码如下:

```
//第 5 章/ reLaunch.vue
methods: {
    choiceT1() {
        this.type1 = 'option'
        uni.reLaunch({
        url: '/pages/index/chat',
            success(res) {
                console.log("成功跳转 chat 页面")
            }
        })
        }
}
```

不过与 navigateTo 方法不同的是，在 HTML5 端调用 uni.reLaunch 之后之前的页面栈会被销毁，但是无法清空浏览器之前的历史记录，此时 navigateBack 不能返回，如果存在历史记录，则当单击浏览器的返回按钮或者调用 history.back()时仍然可以导航到浏览器的其他历史记录。

4. uni.switchTab(OBJECT)

使用该方法会跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面。该方法的参数说明见表 5-5。

表 5-5　uni.switchTab 参数说明

参　数	类　型	是否必填	说　明
url	String	是	需要跳转的 tabBar 页面的路径(需要在 pages.json 的 tabBar 字段定义的页面)且路径后不能带参数
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数(调用成功、失败都会执行)

要使用该方法首先要在 page.json 文件中定义 tabBar。之后在对应的页面编写 switchTab 函数即可，代码如下：

```
    //第 5 章/switchTab.vue
    //在 page.json 文件中定义 tabBar
    {
"tabBar": {
"list": [{
"pagePath": "pages/index/index",
"text": "首页"
},{
"pagePath": "pages/other/chat",
"text": "ChatGPT 功能页"
}]
}
    }
    //在 index 页面中使用 switchTab 跳转
    uni.switchTab({
```

```
        url: '/pages/index/chat
        });
```

5. uni.navigateBack(OBJECT)

使用该方法会关闭当前页面,返回上一页面或多级页面。可通过 getCurrentPages()获取当前的页面栈,决定需要返回几层。该方法的参数说明见表 5-6。

表 5-6 uni.navigateBack 参数说明

参 数	类 型	是否必填	说 明
delta	Number	否	返回的页面数,如果 delta 大于现有页面数,则返回首页
animationType	String	否	窗口关闭的动画效果
animationDuration	Number	否	窗口关闭动画的持续时间,单位为 ms
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数(调用成功、失败都会执行)

该路由函数的使用,代码如下:

```
//第 5 章/uni.switch
//当调用 navigateTo 跳转时,调用该方法的页面会被加入堆栈,而 redirectTo 方法则不会
//此处是 A 页面
uni.navigateTo({
    url: 'B?id = 1'
});
//此处是 B 页面
uni.navigateTo({
    url: 'C?id = 1'
});
//在 C 页面内 navigateBack,将返回 A 页面
uni.navigateBack({
    delta: 2
});
```

以上 5 种方法就是 uni-app 中常用的跳转方法,在 5.4.2 节中将介绍如何在 uni-app 中使用 navigator 组件进行跳转。

5.4.2 使用 navigator 组件进行跳转

使用 navigator 组件进行页面跳转,其效果与 HTML 中的< a >组件类似,但 navigator 只能跳转本地页面,并且目标页面必须在 pages.json 文件中注册。该组件的功能与 API 方式相同,实际上该组件就是将 API 的功能进行了一次封装。在 uni-app 中使用该组件进行页面跳转的代码如下:

```
    //第 5 章/navigator.vue
    < navigator url = "navigate/navigate?title = navigate" hover - class = "navigator - hover">
< button type = "default">跳转到新页面</button >
    </navigator >
```

```
    <navigator url="redirect/redirect?title=redirect" open-type="redirect"
hover-class="other-navigator-hover">
<button type="default">在当前页打开</button>
    </navigator>

    <navigator url="/pages/tabBar/extUI/extUI" open-type="switchTab" hover-class=
"other-navigator-hover">
    <button type="default">跳转 Tab 页面</button>
</navigator>
```

该组件的具体属性说明见表 5-7。

表 5-7　navigator 属性说明

属　性	类　型	默 认 值	说　明
url	String		应用内的跳转链接，值为相对路径或绝对路径
open-type	String	navigate	跳转方式
delta	Number	pop-in/out	当 open-type 取值为'navigateBack'时有效，表示回退的层数
animation-type	String	300	当 open-type 为 navigate、navigateBack 时有效，窗口的显示/关闭动画效果
animation-duration	Number	navigator-hover	当 open-type 为 navigate、navigateBack 时有效，窗口显示/关闭动画的持续时间
hover-class	String	否	指定单击时的样式类，当 hover-class="none"时，没有单击态效果
hover-stop-propagation	Boolean	false	指定是否阻止本节点的祖先节点出现单击态
hover-start-time	Number	50	按住后多久出现单击态，单位为 ms
hover-stay-time	Number	600	手指松开后单击态保留时间，单位为 ms
target	String	self	在哪个小程序目标上发生跳转，默认为当前小程序，值域为 self/miniProgram

其中 open-type 的有效值见表 5-8。

表 5-8　open-type 取值说明

参　数　值	说　明
navigate	对应 uni.navigateTo 的功能
redirect	对应 uni.redirectTo 的功能
switchTab	对应 uni.switchTab 的功能
reLaunch	对应 uni.reLaunch 的功能
navigateBack	对应 uni.navigateBack 的功能
exit	退出小程序，当 target="miniProgram"时生效

可以看到使用 navigator 组件实现页面跳转功能基本和 API 是一致的，只是写法上略有一些区别。navigator 组件在编写代码上更有优势，但是由于组件中没有回调函数属性，所以无法直接调用跳转成功后的失败或者成功的钩子函数，在实际使用中开发者应该根据不同的场景进行选择。

5.5 本章小结

本章首先介绍了如何使用 HBuilder X 软件结合原型图进行页面的布局及代码的编写，通过首页文字显示功能介绍了在 uni-app 中如何使用指令进行数据绑定。之后通过实现首页逐字输出的效果介绍了如何在 uni-app 中使用函数，并通过函数的概念引申出生命周期的概念，并详细地为读者介绍了 uni-app 中生命周期的钩子函数，以及各个钩子函数的使用场景。在最后完成首页跳转功能页的案例中介绍了 uni-app 路由的相关概念，并使用 API 和 navigator 组件的方式介绍了路由跳转。至此首页的功能已经完成，第 6 章将会继续案例项目的功能页开发，并在不断地完善功能的过程中继续为读者介绍 uni-app 中的常用指令及内置方法。