第 5 章 数据存储与访问

随着移动互联网的发展，用户对应用程序的性能、体验等各方面要求都有所增强，虽然现在大多数移动端的数据都由云端（服务端）提供，但是移动端支持离线使用模式也成为用户考虑的重要因素，而离线使用模式必须涉及本地数据的存储与访问机制。本章从用户首选项、文件操作接口模块及关系数据接口模块三方面深入阐述 OpenHarmony 系统的数据存储与访问机制，并结合具体的案例介绍它们的使用方法，以便开发者能够全面地了解它们的原理，更好地开发基于鸿蒙平台的应用。

5.1 概述

操作系统中包含各种各样的数据，按数据结构可分为结构化数据和非结构化数据。能够用统一的数据模型加以描述的数据称为结构化数据，如使用键值对存储和访问的数据、各类数据库数据等；在 OpenHarmony 应用开发中，结构化数据的存储访问由数据管理机制实现。数据结构不规则或不完整的数据、没有预定义数据模型/数据结构的数据称为非结构化数据，如图片、文档、音频和视频文件等；在 OpenHarmony 应用开发中，非结构化数据的存储访问机制由文件管理机制实现。

5.1.1 数据管理机制

数据管理机制为开发者提供了数据存储能力、数据管理能力和数据同步能力。例如，联系人应用中的数据可以保存到数据库中，并提供数据库的安全、可靠以及共享访问等数据管理机制，同时也支持与手表等穿戴设备同步联系人信息。数据存储提供了用户首选项（Preferences）、键值型数据库（KV-Store）和关系数据库（RelationStore）等通用数据持久化能力。数据管理提供了权限管理、数据备份恢复和数据共享框架等高效的数据管理能力。数据同步提供了分布式对象支持内存对象跨设备共享、分布式数据库支持跨设备数据库访问等跨设备数据同步能力。华为官方推出的 API 提供了一系列完整的数据管理模块配套接口，用来实现 OpenHarmony 应用开发中结构化数据的存储与访问。

1. 用户首选项

用户首选项（Preferences）提供了轻量级配置数据的持久化能力，并支持订阅数据变化

的通知能力。数据以文本形式保存在设备中,应用使用过程中会将文本中的数据全部加载到内存,具有访问速度快、效率高的特性,但不适合存储大量数据的场景,不支持分布式同步,常用于保存应用配置信息、用户偏好设置等。

2. 键值型数据库

键值型数据库(KV-Store)提供了键值型数据库的读写、加密、手动备份以及订阅通知能力。它是一种非关系数据库,其数据以“键值”对的形式进行组织、索引和存储,其中,“键”作为唯一标识符。适合很少数据关系和业务关系的业务数据存储,同时因其在分布式场景中降低了解决数据库版本兼容问题的复杂度和数据同步过程中冲突解决的复杂度而被广泛使用。相比于关系数据库,更容易做到跨设备、跨版本兼容。应用需要使用键值型数据库的分布式能力时,KV-Store 会将同步请求发送给 DatamgrService 由其完成跨设备数据同步。

3. 关系数据库

关系数据库提供了关系数据库的增删改查、加密、手动备份以及订阅通知能力。它以行和列的形式存储数据,广泛用于应用中的关系数据的处理,包括一系列的增、删、改、查等接口,开发者也可以运行自己定义的 SQL 语句来满足复杂业务场景的需要。应用需要使用关系数据库的分布式能力时,RelationalStore 部件会将同步请求发送给 DatamgrService 由其完成跨设备数据同步。

5.1.2 文件管理机制

从用户的角度看,文件管理机制为用户提供了按文件名管理文件的能力,即实现“按名存取”。从系统的角度看,文件管理机制是对文件存储设备的空间进行组织和分配,负责文件存储并对存入的文件进行保护和检索的系统。在文件管理模块中,文件系统按文件存储的位置分为本地文件系统和分布式文件系统。本地文件系统提供本地设备或外置存储设备(如 U 盘、移动硬盘)的文件访问能力,本地文件系统是最基本的文件系统;分布式文件系统提供跨设备的文件访问能力。在文件管理模块中,文件按文件的所有者分为应用文件、用户文件和系统文件。应用文件的文件所有者为应用本身,包括应用的安装文件、资源文件和缓存文件等。用户文件的文件所有者为登录到该终端设备的用户,包括用户私有的图片、文档、音频和视频文件等。既不属于应用文件,也不属于用户文件的其他文件就是系统文件,包括公共库、设备文件及系统资源文件等。华为官方推出的 API 提供了一系列完整的文件管理模块配套接口,用来实现 OpenHarmony 应用开发中非结构化数据的存储与访问。

5.2 睡眠质量测试系统的设计与实现

睡眠质量不高与多种疾病有关,其中最突出的就是精神类疾病,睡眠障碍是它的主要表现形式。而大多数人对睡眠认识不够,往往忽略了睡眠与身体健康的关系,这样就造成很多睡眠质量不高的人错失了治疗的最佳时间,因此就需要提供一个方便快捷的方法来帮助这些人了解自身的睡眠状况,让他们尽早得到相应的治疗和帮助。本节以国际公认的睡眠质

量自测量表——阿森斯失眠量表为理论依据，结合 Toggle 组件、Stepper 组件、Stepper Item 组件、页面路由和用户首选项存储与访问机制设计并实现一个睡眠质量测试系统。

5.2.1　Toggle 组件

Toggle(切换)组件用于切换选择状态，包括勾选框样式、状态按钮样式和开关样式，其接口格式如下。

```
1   Toggle(options: { type: ToggleType, isOn?: boolean })
```

(1) type 参数用于设置开关的样式，包括 CheckBox(勾选框样式)、Button(状态按钮样式)和 Switch(开关样式)。

(2) isOn 参数用于设置开关是否打开，默认值为 false(关闭)。

为了满足各种应用开发场景的需要，该组件除支持通用属性和通用事件外，还支持如表 5.1 所示的属性和如表 5.2 所示的事件。

表 5.1　Toggle 组件属性及功能

属　性　名	类　　型	功 能 说 明
selectColor	ResourceColor	设置组件打开状态的背景颜色
switchPointColor	ResourceColor	设置 Switch 类型的圆形滑块颜色(仅 Switch 类型生效)

表 5.2　Toggle 组件事件及功能

事　件　名	功 能 说 明
onChange(callback：(isOn：boolean) ＝＞ void)	状态切换时触发

【例 5-1】　用 Toggle 组件在页面上实现一个模拟开灯、关灯的效果，运行效果如图 5.1 所示。

图 5.1　开关选择器效果

从图 5.1 可以看出,页面由 Column 方式布局的 Image 组件和 Toggle 组件组成,实现代码如下。

```
struct P5_1 {
  @State denResource: string ="/pages/common/black.png"
  build() {
    Column() {
      Image(this.denResource).objectFit(ImageFit.Auto).height("80%")
      Toggle({ type: ToggleType.Switch })
        .onChange((isOn) =>{
            this.denResource = isOn ? "/pages/common/light.png" : "/pages/common/black.png"
        })
        .selectedColor(Color.Red).switchPointColor(Color.Blue)
        .backgroundColor(Color.Orange).borderRadius(8)
    }.width('100%')
  }
}
```

上述第 8 行代码表示如果 isOn 为 true,则在 Image 组件上加载 common 文件夹中的 light.png 图片(代表灯亮),否则加载 common 文件夹中的 black.png 图片(代表灯灭)。

【例 5-2】 设计如图 5.2 所示就业意向调查表,“你在就业过程中最注重什么?”的答案选项由 5 个状态按钮组件组成,但用户只能选其中一项;“你希望从哪些渠道获得企业的招聘信息?”的答案选项由 5 个状态按钮组件组成,用户可以选多项。点击“提交”按钮后,将用户选项结果显示在页面下方。

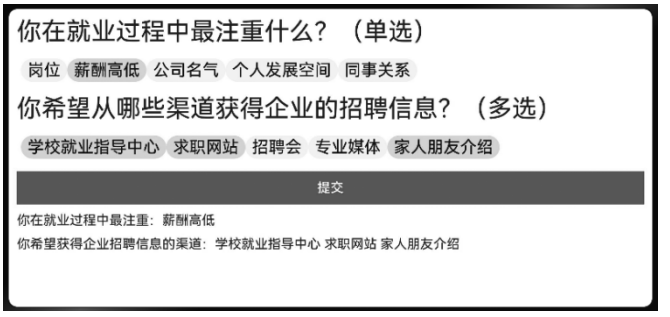

图 5.2 就业意向调查表

从图 5.2 可以看出,整个页面从上至下由就业过程注重区、渠道获取区、按钮区和结果显示区组成,并用 Column 布局方式实现。

就业过程注重区由 Text 组件和 Toggle 组件实现,详细代码如下。

```
@State concern: string[] =['岗位', '薪酬高低', '公司名气', '个人发展空间', '同事关系']
@State concernChecked: boolean[] =[false, false, false, false, false]
//保存注重点选择状态
@State cResult: string =""                    //保存就业注重点
Text("你在就业过程中最注重什么?(单选)").fontSize(30)
```

```
5   Flex({ direction: FlexDirection.Row, wrap: FlexWrap.Wrap }) {
6           ForEach(this.concern, (item: string, index) =>{
7               Toggle ({ type: ToggleType.Button, isOn: this.concernChecked
    [index] }) {
8               Text(item).fontSize(20)   //状态按钮显示文本信息
9             }.onChange((isOn) =>{
10              this.concernChecked[index] =isOn
11              if (isOn) {
12                for (let i =0; i <this.concernChecked.length; i++) {
13                  if (i !=index)  this.concernChecked[i] =false
14                }
15              }
16            })
17          })
18        }.padding(5)
```

上述第 11～15 行表示，如果某个状态按钮处于选中状态，则其他按钮应为未选中状态。同样，渠道获取区也由 Text 组件和 Toggle 组件实现，详细代码如下。

```
1   @State recruit: string[] =['学校就业指导中心', '求职网站', '招聘会', '专业媒
    体', '家人朋友介绍']
2   @State recruitChecked: boolean[] =[false, false, false, false, false]
3   @State rResult: string =""
4   Text("你希望从哪些渠道获得企业的招聘信息?(多选)").fontSize(30)
5   Flex({ direction: FlexDirection.Row, wrap: FlexWrap.Wrap }) {
6           ForEach(this.recruit, (item: string, index) =>{
7               Toggle ({ type: ToggleType.Button, isOn: this.recruitChecked
    [index] }) {
8               Text(item).fontSize(20)
9             }.onChange((isOn)=>{
10              this.recruitChecked[index] =isOn
11            })
12          })
13        }.padding(5)
```

按钮区由 Button 组件实现，结果显示区由两个 Text 组件实现。由于在结果显示区要显示选中状态按钮对应的选项，所以需要首先定义一个自定义函数 showResult()返回选中结果，实现代码如下。

```
1   showResult(msg:string[],checked:boolean[]) :string{
2       let result=""
3       for (let index =0; index <msg.length; index++) {
4         if (checked[index])  result =result +msg[index]+" "
5       }
6       return result
7     }
8   Button("提交").type(ButtonType.Normal).onClick(() =>{
9             this.cResult =this.showResult(this.concern ,this.concernChecked)
10            this.rResult =this.showResult(this.recruit,this.recruitChecked)
11        }).width("100%")
12  Text(`你在就业过程中最注重:${this.cResult}`)
13  Text(`你希望获得企业招聘信息的渠道:${this.rResult}`)
```

5.2.2 Stepper 和 StepperItem 组件

Stepper(步骤导航器)组件用于引导用户完成一个任务需要多个步骤的导航场景,它通常与 StepperItem 组件(步骤导航器子组件)配合使用,StepperItem 组件作为步骤导航器某一个步骤的内容展示组件,是 Stepper 组件的子组件。Stepper 和 StepperItem 的接口格式如下。

```
1  Stepper(value?: { index?: number }){
2    StepperItem(){
3        //子组件
4    }
5  }
```

index 参数用于设置步骤导航器当前显示 StepperItem 的索引值,默认值为 0。

Stepper 组件除支持通用事件外,还支持如表 5.3 所示的事件。StepperItem 组件除支持通用属性外,还支持如表 5.4 所示的属性。

表 5.3 Stepper 组件事件及功能

事 件 名	功 能 说 明
onFinish(callback: () => void)	当步骤导航器最后一个 StepperItem 的 nextLabel 被点击,并且 ItemState 属性为 Normal,触发该回调
onSkip(callback: () => void)	当前显示的 StepperItem 状态为 ItemState.Skip 时,点击 nextLabel 触发该回调
onChange (callback: (prevIndex?: number, index?: number) => void)	点击当前 StepperItem 的 prevLabel 进行步骤切换时触发该回调;或点击当前 StepperItem 的 nextLabel,当前页面不为步骤导航器最后一个 StepperItem 且 ItemState 属性为 Normal 时,触发该回调。prevIndex 表示切换前的步骤页索引值;index 表示切换后的步骤页(前一页或者下一页)索引值
onNext (callback: (index?: number, pendingIndex?: number) => void)	点击 StepperItem 的 nextLabel 切换下一步骤时,当前页面不为步骤导航器最后一个 StepperItem 且 ItemState 属性为 Normal 时,触发该回调。index 表示当前步骤页索引值;pendingIndex 表示下一步骤页索引值
onPrevious(callback: (index?: number, pendingIndex?: number) => void)	点击 StepperItem 的 prevLabel 切换上一步骤时触发该回调。index 表示当前步骤页索引值;pendingIndex 表示上一步骤页索引值

表 5.4 StepperItem 属性及功能

属 性 名	类 型	功 能 说 明
prevLabel	string	设置步骤导航器底部“回退”文本按钮的描述文本
nextLabel	string	设置步骤导航器底部“下一步”文本按钮的描述文本

续表

属　性　名	类　　型	功 能 说 明
status	ItemState	设置步骤导航器 nextLabel 的显示状态，其值包括 ItemState.Normal(默认值，正常状态，右侧文本按钮正常显示，可点击进入下一个 StepperItem)、ItemState.Disabled(不可用状态，右侧文本按钮灰度显示，不可点击进入下一个 StepperItem)、ItemState.Waiting(等待状态，右侧文本按钮不显示，显示等待进度条，不可点击进入下一个 StepperItem)和 ItemState.Skip(跳过状态，右侧文本按钮默认显示“跳过”，可在 Stepper 的 onSkip 事件中定义相关功能)

如果没有定义 prevLabel 属性，在中文语言环境下，默认使用“返回”和“下一步”文本按钮；在非中文语言环境下，默认使用 BACK 和 NEXT 文本按钮；如果是第一个步骤，则页面上没有“回退”文本按钮；如果是最后一个步骤，则页面上的下一步为“开始”文本按钮(中文语言)或者“START”文本按钮(非中文语言)。

【例 5-3】　设计一个如图 5.3 所示的“会员注册”页面，在“会员注册”页面上用步骤导航器分别输入“用户名”“用户密码”和“找回密码问题、找回密码答案”等会员注册信息，点击“提交”按钮，在页面下方显示注册内容。

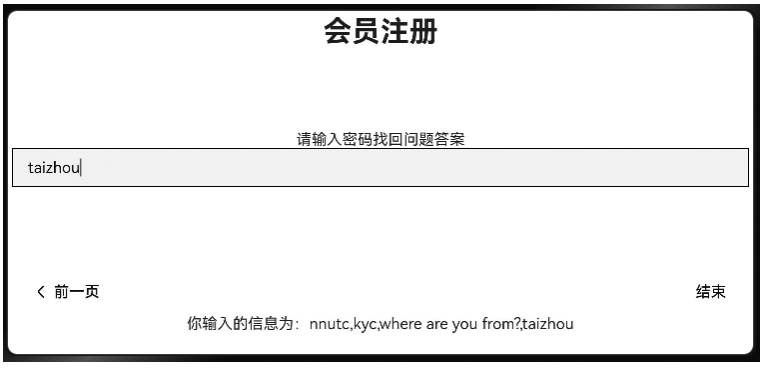

图 5.3　“会员注册”页面

从图 5.3 可以看出，“会员注册”用 Text 组件实现，根据提示输入“用户名”“用户密码”“找回密码问题”和“找回密码答案”等注册信息用 Stepper 和 StepperItem 组件实现，详细代码如下。

```
1   struct P5_3{
2     @State question: string[] =['请输入用户名', '请输入密码', '请输入密码找回问
      题', '请输入密码找回问题答案']                    //注册时回答的问题
3     @State answer: string[] =['', '', '', '']     //注册时输入的问题答案
4     @State index: number =0                       //当前显示 StepperItem 的索引值
5     @State isShow: number =Visibility.None  //控制页面最后一行的注册信息是否显示
6     build() {
7       Column() {
8         Text('会员注册').fontSize(30).fontWeight(FontWeight.Bold)
```

```
9         Stepper() {
10          StepperItem() {                          //第一个问题
11            Column() {
12              Text(this.question[this.index])
13              TextInput().borderRadius(0).borderWidth(1).onChange((value) =>{
14                this.answer[this.index] =value
15              })
16            }
17          }.nextLabel("后一页")
18          StepperItem() {                          //第二个问题
19            //与上述第 11~16 行代码类似,此处略
20          }.prevLabel("前一页").nextLabel("后一页")
21          StepperItem() {                          //第三个问题
22            //与上述第 11~16 行代码类似,此处略
23          }.prevLabel("前一页").nextLabel("后一页")
24          StepperItem() {                          //第四个问题
25            //与上述第 11~16 行代码类似,此处略
26          }.prevLabel("前一页").nextLabel("提交")
27        }
28        .onFinish(() =>{
29          this.isShow =Visibility.Visible
30        })
31        .onChange((preIndex: number, index: number) =>{
32          this.index =index
33        })
34        Text(`你输入的信息为:${this.answer}`).visibility(this.isShow)
35      }.height('80%').padding(5)
36    }
37  }
```

上述第 28～30 行代码表示点击“提交”按钮,将控制页面下方显示答案的第 34 行代码中的 Text 组件的 visibility 属性值设置为 true(显示 Text 组件)。第 31～33 行代码表示 StepperItem 切换时,让保存 question、answer 数组元素下标随之改变。

5.2.3 页面路由

大多数应用程序通常由多个页面组成,并且页面与页面之间可以相互跳转和数据传递,这些能力是由“@ohos.router”模块(页面路由)提供的。在使用“@ohos.router”模块相关功能之前,需要先用如下代码导入该模块。

```
1   import router from '@ohos.router'
```

“@ohos.router”模块提供了一系列接口实现应用内指定页面的跳转、同应用内的某个页面替换当前页面、返回上一页面及返回指定页面等功能。

1. 应用内指定页面的跳转

(1) router.pushUrl(options: RouterOptions): Promise<void>:跳转到应用内的指定页面,结果以 Promise 形式返回。options 参数用于设置跳转页面描述信息,RouterOptions 类型的参数及功能说明如表 5.5 所示;返回值类型为 Promise<void>,表示

异常返回结果，错误码详细说明如表 5.6 所示。

表 5.5　RouterOptions 类型参数及功能说明

参数名	类　型	必　填	功能说明
url	string	是	设置目标页面的 url(例如 pages/index 表示跳转到 index.ets 页面)
params	object	否	设置跳转时要同时传递到目标页面的数据。跳转到目标页面后，使用 router.getParams()获取传递的参数，参数也可以在页面中直接使用，如 this.keyName(keyName 为跳转时 params 参数中的键名值)，如果目标页面中已有该字段，则其值会被传入的字段值覆盖

表 5.6　异常返回结果及说明

错误码	错误信息	结果说明
100001	if UI execution context not found.	可能因获取渲染引擎或解析参数等失败导致内部异常
100002	if the uri is not exist.	路由页面跳转时输入的 URI 错误或不存在
100003	if the pages are pushed too much.	路由压入的页面太多(最多不超过 32)

【例 5-4】　在例 5-3 的基础上，点击页面上的“结束”按钮，跳转到如图 5.4 所示的“注册成功”页面，点击“注册成功”页面上的“返回”按钮，跳转返回到如图 5.3 所示的“会员注册”页面。

图 5.4　“注册成功”页面

由于页面跳转时需要传递用户输入的用户名、密码、找回密码问题及找回密码答案等信息，所以需要自定义 Info 类封装这些信息，Info 类的详细代码如下。

```
1  class Info{
2    userName:string                    //用户名
3    userPwd:string                     //密码
4    userQuestion:string                //找回密码问题
5    userAnswer:string                  //找回密码答案
6  constructor ( userName: string, userPwd: string, userQuestion: string,
   userAnswer:string,) {
7      this.userName =userName
8      this.userPwd =userPwd
9      this.userQuestion =userQuestion
```

```
    this.userAnswer =userAnswer
  }
}
```

根据图 5.4 的显示效果设计“注册成功”页面,页面第一行显示“注册成功”的 Text 组件和显示“返回”按钮的 Button 组件由 Flex 布局实现,其他行信息由 Text 组件实现。在项目的 pages 文件夹下创建 P5_4.ets 页面文件,代码如下。

```
struct P5_4 {
  info: Info =router.getParams() as Info;     //获取传递过来的参数对象并强制转换为
  //Info类型
  build() {
    Column({ space: 5 }) {
      Flex({ justifyContent: FlexAlign.End }) {
        Text("注册成功").width("100%").height("40")
          .textAlign(TextAlign.Center).fontColor(Color.White).fontSize(20)
        Button("返回")
          .type(ButtonType.Normal).width("120").height("40").fontSize(20)
          .onClick(() =>{
            let promise =router.pushUrl({ url: "pages/P5_3" })   //不带参数跳转
            promise.then(() =>{
              console.info("跳转成功")                          //显示跳转成功信息
            }).catch((err: string) =>{
              console.info(`跳转失败!失败信息:${err}`)          //显示跳转失败信息
            })
          })
      }.padding(5).backgroundColor(Color.Gray)
      Text(`您输入的用户名: ${this.info.userName}`).fontSize(25)
      Text(`您输入的密　码: ${this.info.userPwd}`).fontSize(25)
      Text(`您的找回密码问题:${this.info.userQuestion}`).fontSize(25)
      Text(`您的找回密码答案:${this.info.userAnswer}`).fontSize(25)
    }.alignItems(HorizontalAlign.Start).width('100%')
  }
}
```

上述第 2 行代码中的 router.getParams()表示获取发起跳转的页面往当前页传入的参数,该参数的类型为 Object,一般需要将返回的参数进行强制类型转换;上述第 11～16 行代码表示页面路由跳转至“pages/P5_3.ets”页面,并根据跳转结果显示成功与失败信息;如果要显示异常返回结果中的错误码和错误信息,则可以用下述代码替换。

```
try {
        router.pushUrl({ url: "pages/P5_31" })
} catch (err) {
         console.error(`跳转失败, 错误信息代码: ${(err as BusinessError).
code}, 错误信息: ${(err as BusinessError).message}`);
}
```

上述代码中,BusinessError 为 ArkTS 中定义的公共错误信息接口,该接口包含在“@ohos.base”模块中,需要用如下代码导入后才能使用。

```
import { BusinessError } from '@ohos.base'
```

最后,在例 5-3 实现代码的第 29 行下面添加如下代码即可实现页面的跳转。