第 5 章

数据存储与访问

数据存储是开发应用程序时需要解决的最基本问题，虽然客户端 App 的数据一般都由云端（服务端）提供，但是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，目前客户端 App 一般都需要支持离线使用模式，因此在进行客户端应用程序开发时需要客户端实现本地数据的存储与访问。本章结合具体案例介绍方舟开发框架提供的 JS API 实现本地存储与访问数据的机制，以便读者全面地了解本地数据存储与访问接口，更好地开发 HarmonyOS 应用程序。

扫一扫

5.1 概述

HarmonyOS 应用程序开发的数据管理机制包括轻量级数据存储（偏好文件）、文件、关系数据库、对象关系映射数据库、分布式数据服务和分布式文件服务等。华为官方推出的 JS API 提供了一系列完整的配套接口，用来实现 HarmonyOS 应用程序开发中的数据存储和访问机制。读者需要注意，JS API 中的部分数据存储与访问接口从 API version 6 开始华为官方不再维护，所以推荐使用官方提供的最新接口开发 HarmonyOS 应用程序。

5.1.1 轻量级数据存储与访问机制

在 HarmonyOS 应用程序开发中，轻量级数据存储以 key-value 结构的形式对数据进行存取操作。数据存储形式为键值对，键（key）是不重复的关键字，它的类型为字符串型；值（value）是数据值，它存储数据的类型包括数字型、字符型、布尔型。应用程序在获取某个轻量级数据存储对象后，该存储对象中的数据将会被缓存在内存中，以便应用程序以更快的速度实现数据访问，提高效率。当然，应用程序也可以将缓存的数据写入 HarmonyOS 终端设备存储器的文本文件中进行持久化存储。由于文件读写会产生不可避免的系统资源开销，因此开发者在开发应用程序时，应尽量减少对存放在 HarmonyOS 终端设备存储器中文本文件的读写频率。轻量级数据存储访问机制适用于应用程序在本地存储少量数据，经常应用于操作键值对形式数据的场景。从 API version 6 开始，JS API 提供的@ohos.data.storage 接口可以实现数据存储对象的获取和写入。

5.1.2 文件存储与访问机制

文件通常应用于将数据以普通文件格式下载或保存到 HarmonyOS 终端设备的本地存储空间。从 API version 6 开始，JS API 提供的@ohos.fileio 接口可以实现对文件或目录进行操作。但是，在操作文件或目录前，必须先通过 Ability 上下文提供的相关接口，获取操作对象在内部存储目录或内部存储缓存目录的绝对路径，然后才能进行文件或目录的创建、打开、读取、写入、复制、删除、重命名、修改操作权限、修改所有者等操作。

5.1.3 关系数据库存储与访问机制

关系数据库（Relational Database，RDB）是一种基于关系模型管理数据的数据库。HarmonyOS 关系数据库基于 SQLite 组件提供了一套完整的对本地数据库进行管理的机制，对外提供了一系列的增、删、改、查等接口，也可以直接运行用户输入的 SQL（Structured Query Language，结构化查询语言）语句来满足复杂的场景需要。SQLite 是一款轻量级的关系数据库管理系统，它既支持事务和批处理、自动版本管理、标准的 CURD（Create、Update、Retrieve、Delete）操作，也遵守 ACID（Atomic、Consistency、Isolation、Durability）特性。HarmonyOS 提供的关系数据库功能更加完善，查询效率更加高效，从 API version 7 开始，JS API 提供的@ohos.data.rdb 接口可以实现关系数据的操作。

5.1.4 对象关系映射数据库存储与访问机制

对象关系映射（Object Relational Mapping，ORM）数据库是一款基于 SQLite 的数据库框架，屏蔽了底层 SQLite 数据库的 SQL 操作，针对实体和关系提供了增、删、改、查等一系列的面向对象接口。在进行 HarmonyOS 应用程序开发时，开发者不必再编写复杂的 SQL 语句，而是直接以操作对象的形式操作数据库，所以在提升开发效率的同时，也能让开发者聚焦于业务开发。对象关系映射数据库跟关系数据库一样，都使用 SQLite 作为持久化引擎，底层使用的是同一套数据库连接池和数据库连接机制。对象关系映射数据库适用于开发者使用的数据可以分解为一个或多个对象，且需要对数据进行增、删、改、查等操作，但是不希望编写过于复杂的 SQL 语句的场景。

5.2 睡眠质量测试系统的设计与实现

如今睡眠已经成为全世界人们共同关注的问题。由于长期睡眠质量不高，常会伴随着多种疾病的产生，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测量自己的睡眠状况，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 switch 组件、stepper 组件、stepper-item 组件、页面路由和

轻量级数据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　switch 组件


扫一扫

switch 组件(开关选择器组件)用于开启或关闭某个功能,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　switch 组件属性及功能

属性名	类型	功 能 说 明
checked	boolean	设置开关是否选中,属性值包括 false(默认值,未选中)和 true
showtext	boolean	设置开关是否显示文本,属性值包括 false(默认值,不显示)和 true
texton	string	设置开关选中时显示的文本,默认值为 On
textoff	string	设置开关选中时显示的文本,默认值为 Off

表 5.2　switch 组件事件及功能

事件名	返　回　值	功 能 说 明
change	{ checked: checkedValue }	开关选择器选中状态改变时,触发该回调事件

【范例 5-1】　用 switch 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

css 的代码如下。

```
.container {
    display: flex;
    flex-direction: column;
    align-items: center;
    margin: 15fp;
    width: 100%;
    height: 100%;
}
.switch-css {
    texton-color: blue;
    textoff-color: red;
    font-size: 25fp;
}
```

上述第 9～13 行代码定义 switch 组件的样式，texton-color 样式表示开关选择器处于选中状态时显示的文本颜色，textoff-color 样式表示开关选择器处于未选中状态时显示的文本颜色。

hml 的代码如下。

```
<div class="container">
    <image style="height:90%;object-fit:fill;" src="{{imgSrc}}"></image>
    <switch class="switch-css" showtext="true" checked="{{isChecked}}"
@change="controlLamp"></switch>
</div>
```

上述第 2 行代码用 image 组件加载代表灯亮的图片(light.png)和代表灯灭的图片(black.png)，这两张图片需要先复制到项目的 common/images 文件夹中。

js 的代码如下。

```
export default {
    data: {
        imgSrc:'/common/images/black.png',          //默认加载代表灯灭的图片
        isChecked:false                             //默认开关选择器未选中
    },
    controlLamp(e){
        if(e.checked){
            this.imgSrc = '/common/images/light.png'//加载代表灯亮的图片
        }else{
            this.imgSrc = '/common/images/black.png'//加载代表灯灭的图片
        }
    }
}
```

上述第 7～11 行代码表示如果开关选择器处于选中状态，则加载代表灯亮的图片文件(light.png)，否则加载代表灯灭的图片文件(black.png)。

5.2.2　轻量级数据存储与访问接口

轻量级数据存储为应用程序提供 key-value(键值对)类型的文件数据处理能力，支持应用程序对数据进行轻量级的存储及查询。数据存储形式为键值对，键的类型为字符串型，最大长度为 80B；值的存储数据类型包括数字型、字符型和布尔型，其中字符型值的最大长度为 8192B。在 HarmonyOS 应用程序开发中，借助 JS API 提供的@ohos.data.storage 接口可以实现数据存储对象的获取和写入，写入的数据保存在 HarmonyOS 终端设备存储器的文本文件中。

1. 读取指定文件


扫一扫

- dataStorage.getStorageSync(path：string)：Storage，同步读取指定文件，并将数据加载到 Storage 类型的实例。getStorageSync 参数及功能说明如表 5.3 所示。getStorageSync 返回值类型及功能说明如表 5.4 所示。

表 5.3　getStorageSync 参数及功能说明

参数名	类型	必填	功 能 说 明
path	string	是	设置应用程序内部数据存储路径

表 5.4　getStorageSync 返回值类型及功能说明

返回值类型	功 能 说 明
Storage	返回 Storage 类型实例，可用于数据存储及查询操作

【范例 5-2】　用同步方式读取应用程序内部数据存储路径下的 config.ini 文件，并将 config.ini 文件的存储路径显示在页面上，运行效果如图 5.2 所示。

图 5.2　同步读取文件目录路径

hml 的代码如下。

```
<div class="container">
    <text >config.ini 文件的存储路径: {{ pathName }}</text>
    <button type="capsule" @click="openConfig">打开文件</button>
</div>
```

上述第 1 行代码引用的 container 样式类代码内容与范例 5-1 的 container 样式类代码完全一样，限于篇幅，这里不再赘述。

js 的代码如下。

```
import dataStorage from '@ohos.data.storage'
import featureAbility from '@ohos.ability.featureAbility'
export default {
    data: {
        pathName: "                          //保存 config.ini 文件的存放目录
        storage:"                            //保存数据存储实例
    },
    async openConfig() {
        var context = featureAbility.getContext()  //获取上下文
        var path = await context.getFilesDir()     //获得应用程序内部存储目录
        this.storage = dataStorage.getStorageSync(path + '/config.ini')
                                             //读取 config.ini 文件
        this.pathName = path
        //操作 config.ini 文件中的数据
     }
}
```

HarmonyOS 应用程序使用轻量级数据存储与访问机制存取数据时，必须首先从@ohos.data.storage 接口中导入 dataStorage 包，即上述第 1 行代码；然后引用该包中的各种方法对轻量级数据进行操作。轻量级数据存储与访问实际上就是对内部存储器上以键值对格式存储的文本文件进行操作，所以在读取文件时必须指明文件的存放位置，即上述第 2 行及 9～10 行代码，其中第 2 行代码表示从@ohos.ability.featureAbility 接口中导入 featureAbility 包，第 9～10 行代码表示调用该包中的 getContext()方法获得 Ability 上下文；再通过 Ability 上下文调用 getFilesDir()方法异步获取应用程序在内部存储器上的文件路径。由于 getFilesDir()方法是异步(async)执行的，所以上述第 8～14 行代码定义了一个异步事件，其中第 10 行代码中的 await 表示只有在获得应用程序内部存储目录后，才能执行第 11 行及后面的代码。虽然异步事件会返回一个 Promise 对象，但代码前用 await 关键字表示可以等待异步事件的返回值，也就是说，await 不仅可以用于等待返回 Promise 对象，而且它还可以等待任意表达式的结果，所以上述第 10 行代码直接将返回的值赋给 path 变量。从图 5.2 可以看出，getFilesDir()方法返回的应用程序在内部存储器上的文件路径为“/data/data/应用程序包名/files”。如果将上述第 10 行代码修改为下述代码，则表示获取应用程序内部存储器上的缓存目录路径，运行效果如图 5.3 所示。getCacheDir()方法返回

的应用程序在内部存储器上的缓存目录路径为“/data/data/应用程序包名/cache”。

图 5.3　同步读取缓存目录路径

```
1    var path = await context.getCacheDir()      //获得应用程序内部存储缓存目录
```

- dataStorage.getStorage(path：string，callback：AsyncCallback<Storage>)：void，异步读取指定文件，并将数据加载到 Storage 类型的实例，以 callback 形式返回结果。getStorage 参数及功能说明如表 5.5 所示。

表 5.5　getStorage 参数及功能说明

参数名	类　型	必填	功 能 说 明
path	string	是	设置应用程序内部数据存储路径
callback	AsyncCallback<Storage>	是	回调函数

扫一扫

例如，要实现范例 5-2 的功能，也可以将范例 5-2 的第 11～13 行 js 代码修改为如下代码。

```
1          var that = this
2          dataStorage.getStorage(path + '/config.ini', function (err, storage) {
3              if (err) {
4                  that.pathName ="读内部存储文件失败"
```

```
                return;
            }
            //用 storage 实例操作 config.ini 文件中的数据
            that.pathName = path
        })
```

- dataStorage.getStorage(path: string): Promise<Storage>,异步读取指定文件,并将数据加载到 Storage 实例。getStorage 参数及功能说明如表 5.3 所示。getStorage 返回值类型及功能说明如表 5.6 所示。

表 5.6 getStorage 返回值类型及功能说明

返回值类型	功能说明
Promise<Storage>	返回 Promise 实例,用于异步获取结果

例如,要实现范例 5-2 的功能,也可以将范例 5-2 的第 11～13 行 js 代码修改为如下代码。

```
var that = this
let promise = dataStorage.getStorage(path + '/config.ini')
promise.then((storage) => {
    //用 storage 实例操作 config.ini 文件中的数据
    that.pathName = path
}).catch((err) => {
    that.pathName = "读内部存储文件失败"
})
```

2. 向指定存储对象写数据


扫一扫

- putSync(key: string, value: ValueType): void,同步将数据写入 Storage 类型的实例,并用 flush()或 flushSync()方法将 Storage 类型的实例写入指定文件保存。putSync 参数及功能说明如表 5.7 所示。

表 5.7 putSync 参数及功能说明

参数名	类　型	必填	功能说明
key	string	是	设置要存储或修改的键,不能为空
value	number、string 或 boolean	是	设置要存储或修改的值

【范例 5-3】 用同步方式将登录用户名(key 为 loginName,value 为 nipaopao)和登录密码(key 为 loginPwd,value 为 123456)以键值对方式保存在应用程序内部存储路径下的 config.ini 文件中。

在实现范例 5-2 的基础上,在 hml 的代码中增加 1 个“保存数据”按钮,其代码如下。

```
<button type="capsule" @click="writeConfig">保存数据</button>
```

在实现范例 5-2 的基础上，在 js 的代码中增加 1 个“保存数据”单击事件的回调方法，其代码如下。

```
writeConfig() {
    this.storage.putSync('loginName', 'nipaopao')
                                        //key 为'loginName',value 为'nipaopao'
    this.storage.putSync('loginPwd', '123456')
                                        //key 为'loginPwd',value 为'123456'
    this.storage.flushSync()            //保存到文件中
    console.info("保存成功")
}
```

- put(key：string，value：ValueType，callback：AsyncCallback＜void＞)：void，异步将数据写入 Storage 类型的实例，并用 flush()或 flushSync()方法将 Storage 类型的实例写入指定文件保存。put 参数及功能说明如表 5.8 所示。


扫一扫

表 5.8　put 参数及功能说明

参数名	类　　型	必填	功 能 说 明
key	string	是	设置要存储或修改的键，不能为空
value	number、string 或 boolean	是	设置要存储或修改的值
callback	AsyncCallback＜Storage＞	是	回调函数

例如，要实现范例 5-3 的功能，也可以将“保存数据”单击事件的回调方法修改为如下代码。

```
writeConfig(){
    var that = this
    this.storage.put('loginName', 'nipaopao', function (err) {
                                                //保存登录用户名
        if (err) {
            console.info("保存失败:" + err)
            return
        }
        that.storage.flushSync()
        console.info("保存成功")
    })
    //保存登录密码的代码类似，此处略
}
```

- put(key：string，value：ValueType)：Promise＜void＞，异步将数据写入 Storage 类型的实例，并用 flush()或 flushSync()方法将 Storage 类型的实例写入指定文件保存。put 参数及功能说明如表 5.8 所示。put 返回值类型及功能说明如表 5.9 所示。

表 5.9 put 返回值类型及功能说明

返回值类型	功 能 说 明
Promise<void>	返回 Promise 实例，用于异步处理

例如，要实现范例 5-3 的功能，也可以将“保存数据”单击事件的回调方法修改为如下代码。

```
writeConfig() {
        let promise = this.storage.put('loginName', 'nipaopao')
        promise.then(() => {
            console.info("保存成功")
        }).catch((err) => {
            console.info("保存失败:" + err)
        })
        //保存登录密码的代码类似，此处略
}
```

扫一扫

3. 从指定存储对象中读数据

- getSync(key：string，defValue：ValueType)：ValueType，同步获取指定键的值，如果值为 null 或者非默认值类型，则返回默认数据。getSync 参数及功能说明如表 5.10 所示。

表 5.10 getSync 参数及功能说明

参数名	类　　型	必填	功 能 说 明
key	string	是	设置要获取的键，不能为空
defValue	number、string 或 boolean	是	若给定的键不存在，则设置该键返回的默认值

【范例 5-4】 用同步方式从应用程序内部存储路径下的 config.ini 文件中读出登录用户名(key 为 loginName)和登录密码(key 为 loginPwd)，并显示在如图 5.4 所示页面的对应位置。

图 5.4 同步读出轻量级数据