第5章

鸿蒙中的数据管理

5.1 本章简介

本章将系统介绍 HarmonyOS 在数据管理方面的核心能力，重点围绕 ArkData（方舟数据管理）框架下的多样化存储方案与分布式数据协同机制展开。首先讲解用户首选项的轻量级数据持久化实现，涵盖键值存储、数据刷盘及内存管理机制；其次深入探讨关系数据库的设计原理、接口调用及实际应用，通过建表、事务、索引及增删改查操作演示本地数据持久化流程。此外，本章还结合实际项目案例“豹考通高考信息提醒”，完整展示 HarmonyOS 中 SQLite 数据库的集成、沙箱路径管理及数据查询与渲染的全流程，帮助读者掌握在复杂业务场景下实现数据存储与同步的开发技能。通过本章的学习，读者将能够深入理解 HarmonyOS 在数据管理方面的高效性、安全性与多端协同能力，为构建具备数据持久化与跨设备同步功能的应用奠定坚实基础。

5.2 方舟数据管理

ArkData 简介

数据管理是利用计算机硬件和软件技术对数据进行有效的收集、存储、处理和应用的过程。其目的在于充分有效地发挥数据的作用。实现数据有效管理的关键是数据组织。

随着计算机技术的发展，数据管理经历了人工管理、文件系统、数据库系统三个发展阶段。在数据库系统中所建立的数据结构，更充分地描述了数据间的内在联系，便于数据修改、更新与扩充，同时保证了数据的独立性、可靠、安全性与完整性，减少了数据冗余，故提高了数据共享程度及数据管理效率。

ArkData（方舟数据管理）为开发者提供数据存储、数据管理和数据同步能力，如联系人应用数据可以保存到数据库中，提供数据库的安全、可靠以及共享访问等管理机制，也支持与手表同步联系人信息。

与安卓开发相比，鸿蒙开发通常使用 SharedPreferences 进行轻量级数据存储，以及 SQLite 数据库进行更复杂的数据存储。HarmonyOS 的存储机制在提供类似功能的同时，强调了数据在不同组件间的共享和同步，特别是跨设备的同步能力。

总的来说，HarmonyOS 的 ArkData 在状态管理和数据持久化方面提供了更加丰富和灵活的选项，特别是对于应用级和页面级的状态共享，以及 UI 的紧密集成。而安卓则更多依赖于传统的文件存储和数据库存储方式。

5.3　用户首选项实现数据持久化

用户首选项为应用提供 Key-Value 键值型的数据处理能力，支持应用持久化轻量级数据，并对其修改和查询。当用户希望有一个全局唯一存储的地方，可以采用用户首选项来进行存储。Preferences 会将该数据缓存在内存中，当用户读取的时候，能够快速从内存中获取数据，当需要持久化时可以使用 flush 接口将内存中的数据写入持久化文件中。Preferences 会随着存放的数据量越多而导致应用占用的内存越大，因此，Preferences 不适合存放过多的数据，也不支持通过配置加密，适用的场景一般为应用保存用户的个性化设置。用户首选项功能示意图如图 5-1 所示。

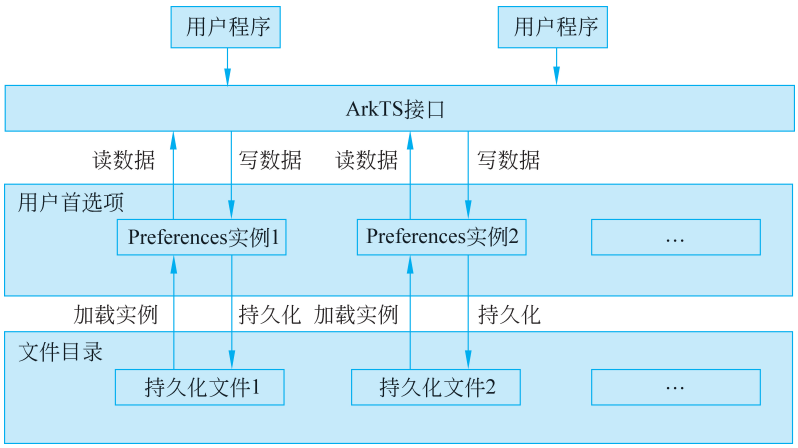

图 5-1　用户首选项功能示意图

用户程序通过 ArkTS 接口调用用户首选项读写对应的数据文件。开发者可以将用户首选项持久化文件的内容加载到 Preferences 实例，每个文件唯一对应一个 Preferences 实例，系统会通过静态容器将该实例存储在内存中，直到主动从内存中移除该实例或者删除该文件。

5.3.1　接口说明

如表 5-1 所示是用户首选项持久化功能的相关接口，更多的接口详见开发者文档。

表 5-1　用户首选项相关接口表

接口名称	描　述
getPreferencesSync(context: Context, options: Options): Preferences	获取 Preferences 实例。该接口存在异步接口
putSync(key: string, value: ValueType): void	将数据写入 Preferences 实例，可通过 flush 将 Preferences 实例持久化。该接口存在异步接口
hasSync(key: string): boolean	检查 Preferences 实例是否包含名为给定 Key 的存储键-值对。给定的 Key 值不能为空。该接口存在异步接口
getSync(key: string, defValue: ValueType): ValueType	获取键对应的值，如果值为 null 或者非默认值类型，返回默认数据 defValue。该接口存在异步接口

续表

接 口 名 称	描　　述
deleteSync(key: string): void	从 Preferences 实例中删除名为给定 Key 的存储键-值对。该接口存在异步接口
flush(callback:AsyncCallback<void>): void	将当前 Preferences 实例的数据异步存储到用户首选项持久化文件中
on(type:'change',callback:Callback<string>): void	订阅数据变更,订阅的数据发生变更后,在执行 flush 方法后,触发 callback 回调
deletePreferences (context: Context, options: Options,callback: AsyncCallback<void>): void	从内存中移除指定的 Preferences 实例。若 Preferences 实例有对应的持久化文件,则同时删除其持久化文件

5.3.2 使用步骤

首先,需要创建一个 ArkTS 文件 src/main/ets/common/PreferencesUtils.ts,存放用户首选项的方法类如程序清单 5-1 所示。

程序清单 5-1　Code050302\MyApplication_53\entry\src\main\ets\common\PreferencesUtils.ts

```
import dataPreferences from '@ohos.data.preferences'
import hilog from '@ohos.hilog';

/**
 * 用户首选项(存储简单数据)
 */
export default class PreferencesUtil {
  //用户首选项名称
  private static preferenceName: string = 'myStore'

  /**
   * 创建
   * @param context
   */
  static createPreferences(context) {
    globalThis.getFontPreferences = (() => {
      let preferences: Promise<dataPreferences.Preferences> =
        dataPreferences.getPreferences(context, this.preferenceName);
      return preferences;
    });
  }

  /**
   * 存放数据
   * @param value
   */
  static savePreferencesValue(key: string, value: string) {
    globalThis.getFontPreferences().then((preferences) => {
      preferences.has(key).then(async (isExist) => {
        if (! isExist) {
          await preferences.put(key, value);
          preferences.flush();
        }
```

```
34          }).catch((err) => {
35              hilog.info(0xFF00, 'preferencesTag', '%{public}s', 'save
                PreferencesValue fail');
36          });
37        }).catch((err) => {
38          hilog.info(0xFF00, 'preferencesTag', '%{public}s', 'save PreferencesValue
            fail');
39        });
40      }
41
42      /**
43       * 获取数据
44       * @returns
45       */
46      static async getPreferencesValue(key: string) {
47        let value: string = '';
48        const preferences = await globalThis.getFontPreferences();
49        value = await preferences.get(key, value);
50        return value;
51      }
52
53      /**
54       * 删除数据
55       */
56      static async deletePreferencesValue(key: string) {
57          const preferences: dataPreferences.Preferences = await globalThis.
            getFontPreferences();
58        let deleteValue = preferences.delete(key);
59        deleteValue.then(() => {
60              hilog.info(0xFF00, 'preferencesTag', '%{public}s', 'delete
                PreferencesValue success');
61        }).catch((err) => {
62              hilog.info(0xFF00, 'preferencesTag', '%{public}s', 'delete
                PreferencesValue fail');
63        });
64      }65     }
```

然后，需要导入@kit.ArkData 模块 import { preferences } from '@kit.ArkData'，在 src/main/ets/entryability/EntryAbility.ets 中的 onCreate 函数中初始化用户首选项，代码如程序清单 5-2 所示。

程序清单 5-2　Code050302\MyApplication_53\entry\src\mainets\entryability\EntryAbility.ets

```
1   onCreate(want: Want, launchParam: AbilityConstant.LaunchParam): void {
2     hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onCreate');
3     PreferencesUtil.createPreferences(this.context)
4   }
```

最后，需要在 pages 中编写一个简单的 UI，使用用户首选项的这种方法，就可以通过 PreferencesUtil 工具类中的 getPreferencesValue()方法，调用 preferences 工具类中的一系列方法。

具体代码如程序清单 5-3 所示。

程序清单 5-3　Code050302\MyApplication_53\entry\src\mainets\pages\index.ets

```
import PreferencesUtil from '../common/PreferencesUtils'

@Entry
@Component
struct Index {

  @State labname:string=""
  build() {
    Column() {
      Text('添加数据')
        .onClick(() => {
          PreferencesUtil.savePreferencesValue('labname', '倚动实验室')
          console.log('preferences save success')
        }).margin({bottom:50})

      Divider()

      Text('查看数据:  '+this.labname)
        .onClick(async () => {
          this.labname = await PreferencesUtil.getPreferencesValue
          ('labname')
            console.log('preferences username ' + JSON.stringify(this.
            labname))
        }).margin({bottom:50})

      Divider()

      Text('删除数据')
        .onClick(async () => {
          await PreferencesUtil.deletePreferencesValue('labname')
          console.log('preferences remove success')
        })
    }
    .height('100%')
    .width('100%')
  }
}
```

代码部分完成后,需要打开模拟器,并运行该程序,就可以得到图 5-2 所示的界面,单击“添加数据”文本框,就可以响应 onClick()方法,通过 PreferencesUtil 工具类中的 savePreferencesValue()方法,以 flush 刷盘的方式将数据存储到用户首选项持久化文件中。

然后单击“查看数据”文本框接口 preferences.get(key,value);通过键对匹配找到 key 值所对应的数据。

当退出应用、关机重启时,其所存储的数据不会因为模拟器的关机而删除,而是持久地保存在了文件中,如图 5-3 所示。再次进入应用时单击“查看数据”文本框,依然能显示原先所存储的数据。

最后，单击“删除数据”文本框，通过调用 PreferencesUtil 工具类中的 deletePreferencesValue() 方法，进而通过 preferences 中的接口 preferences.delete(key)完成删除操作，操作完成后，单击“查看数据”，刚刚存入的文本内容置空，数据就已经删除了。

图 5-2　用户首选项功能示意图

图 5-3　数据持久保存

5.3.3　数据查看

所存储的数据文件可以在 Device File Browser 中查看，当看到图 5-4 所示的包文件 data/app/el2/100/base/包名/haps/entry/preferences 时，则代表用户首选项的数据已经得到，如图 5-5 所示。

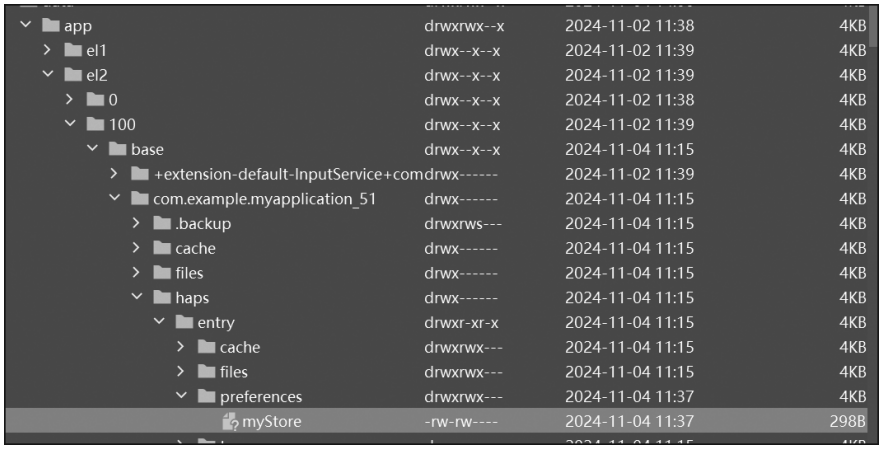

图 5-4　Device File Browser

EntryAbility.ets | PreferencesUtils.ts | Preferences.ets | Index.ets | myStore | EntryBackupAbility.ets

```
<?xml version="1.0" encoding="UTF-8"?>
<preferences version="1.0">
  <string key="labname">倚动实验室</string>
  <string key="startup">auto</string>
  <uint8Array key="uInt8">fu+8gUAj77+lJeKApuKApiYq77yI77yJ4oCU4oCUK++8nw==</uint8Array>
  <string key="username">东林</string>
</preferences>
```

图 5-5　myStore 文件

5.4　关系数据库实现数据持久化

关系数据库(Relational Database,RDB)是指采用了关系模型来组织数据的数据库,是创建在关系模型基础上的数据库,以行和列的形式存储数据,方便用户理解。关系数据库可以理解为由二维表及其之间的关系组成的一个数据组织。

HarmonyOS 关系数据库是基于 SQLite 组件实现的,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、查接口,也可以直接运行开发者输入的 SQL 语句来满足复杂的场景需要。关系数据库为应用提供通用的操作接口,底层使用 SQLite 作为持久化存储引擎,支持 SQLite 具有的数据库特性,包括但不限于事务、索引、视图、触发器、外键、参数化查询和预编译 SQL 语句。关系数据库示意图如图 5-6 所示。

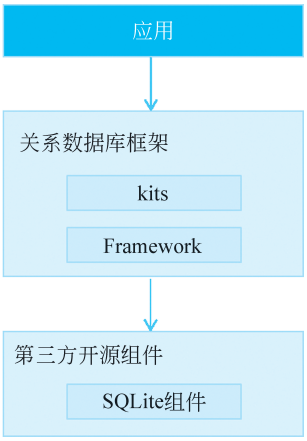

图 5-6　关系数据库示意图

5.4.1　接口说明

以下是关系数据库持久化功能的相关接口,大部分为异步接口。异步接口均有 callback 和 promise 两种返回形式,表 5-2 以 callback 形式为例,列出了常见的接口说明,更多接口及使用方式详见开发者文档。

表 5-2　用户首选项功能示意图

接 口 名 称	描　　述
getRdbStore (context: Context, config: StoreConfig, callback: AsyncCallback<RdbStore>): void	获得一个 RdbStore,操作关系数据库,用户可以根据自己的需求配置 RdbStore 的参数,然后通过 RdbStore 调用相关接口可以执行相关的数据操作
executeSql(sql: string,bindArgs: Array<ValueType>, callback: AsyncCallback<void>):void	执行包含指定参数但不返回值的 SQL 语句

续表

接 口 名 称	描　　述
insert（table：string，values：ValuesBucket，callback：AsyncCallback<number>）：void	向目标表中插入一行数据
update(values：ValuesBucket，predicates：RdbPredicates，callback：AsyncCallback<number>)：void	根据 predicates 的指定实例对象更新数据库中的数据
delete(predicates：RdbPredicates，callback：AsyncCallback<number>)：void	根据 predicates 的指定实例对象从数据库中删除数据
query(predicates：RdbPredicates，columns：Array<string>，callback：AsyncCallback<ResultSet>)：void	根据指定条件查询数据库中的数据
deleteRdbStore(context：Context，name：string，callback：AsyncCallback<void>)：void	删除数据库

5.4.2　使用步骤

使用关系数据库实现数据持久化，需要获取一个 RdbStore，其中包括建库、建表、升降级等操作。示例代码如程序清单 5-4 所示。

程序清单 5-4　Code050402\MyApplication_54\entry\src\main\ets\entryability\EntryAbility.ets

```
onWindowStageCreate(windowStage: window.WindowStage): void {
    //Main window is created, set main page for this ability
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onWindowStageCreate');

    windowStage.loadContent('pages/Index', (err) => {
      if (err.code) {
        hilog.error(0x0000, 'testTag', 'Failed to load the content. Cause:
        %{public}s', JSON.stringify(err) ?? '');
        return;
      }
      hilog.info(0x0000, 'testTag', 'Succeeded in loading the content.');
    });

    const STORE_CONFIG :relationalStore.StoreConfig= {
      name: 'RdbTest1.db', //数据库文件名
      securityLevel: relationalStore.SecurityLevel.S3, //数据库安全级别
      encrypt: false, //可选参数,指定数据库是否加密,默认不加密
      customDir: 'customDir/subCustomDir',   //可选参数,数据库自定义路径。
      //数据库将在如下目录结构中被创建: context.databaseDir + '/rdb/' +
      //customDir,其中 context.databaseDir 是应用沙箱对应的路径,'/rdb/'表
      //示创建的是关系数据库,customDir 表示自定义的路径。当此参数不填时,
      //默认在本应用沙箱目录下创建 RdbStore 实例。
      isReadOnly: false  //可选参数,指定数据库是否以只读方式打开。该参数默认
      //为 false,表示数据库可读可写。该参数为 true 时,只允许从数据库读取数据,
      //不允许对数据库进行写操作,否则会返回错误码 801。
    };

    //判断数据库版本,如果不匹配则需进行升降级操作
```

```
22        //假设当前数据库版本为 3,表结构: EMPLOYEE (NAME, AGE, SALARY, CODES, IDENTITY)
23          const SQL_CREATE_TABLE = 'CREATE TABLE IF NOT EXISTS EMPLOYEE (ID
            INTEGER PRIMARY KEY AUTOINCREMENT, NAME TEXT NOT NULL, AGE INTEGER,
            SALARY REAL, CODES BLOB, IDENTITY UNLIMITED INT)';   //建表 SQL 语句,
            //IDENTITY 为 bigint 类型,SQL 中指定类型为 UNLIMITED INT
24
25        relationalStore.getRdbStore(this.context, STORE_CONFIG, (err, store) => {
26          if (err) {
27            console.error('Failed to get RdbStore. Code:${err.code},
              message:${err.message}');
28            return;
29          }
30          console.info('Succeeded in getting RdbStore.');
31
32          //当数据库创建时,数据库默认版本为 0
33          if (store.version === 0) {
34            store.executeSql(SQL_CREATE_TABLE);              //创建数据表
35            //设置数据库的版本,入参为大于 0 的整数
36            store.version = 3;
37          }
38
39          //如果数据库版本不为 0 且和当前数据库版本不匹配,需要进行升降级操作
40          //当数据库存在并假定版本为 1 时,则应用从某一版本升级到当前版本,数据库需要
            //从 1 版本升级到 2 版本
41          if (store.version === 1) {
42            //version = 1: 表结构: EMPLOYEE (NAME, SALARY, CODES, ADDRESS) =>
              //version = 2: 表结构: EMPLOYEE (NAME, AGE, SALARY, CODES, ADDRESS)
43            (store as relationalStore.RdbStore).executeSql('ALTER TABLE
              EMPLOYEE ADD COLUMN AGE INTEGER');
44            store.version = 2;
45          }
46
47          //当数据库存在并假定版本为 2 时,则应用从某一版本升级到当前版本,数据库需要
            //从 2 版本升级到 3 版本
48          if (store.version === 2) {
49            //version = 2: 表结构: EMPLOYEE (NAME, AGE, SALARY, CODES, ADDRESS)
              //=> version = 3: 表结构: EMPLOYEE (NAME, AGE, SALARY, CODES)
50            (store as relationalStore.RdbStore).executeSql('ALTER TABLE
              EMPLOYEE DROP COLUMN ADDRESS TEXT');
51            store.version = 3;
52          }
53        });
54      }
```

然后在 pages/index.ets 中输入下列代码,定义一个 store 和一个 getRdbStore 函数,并对其中的关系数据库进行配置,代码如程序清单 5-5 所示。

程序清单 5-5　Code050402\MyApplication_54\entry\src\main\ets\pages\index.ets

```
1   store: relationalStore.RdbStore | undefined = undefined;
2     getRdbStore() {
3       const STORE_CONFIG: relationalStore.StoreConfig = {
4         name: "RdbTest53.db",
5         securityLevel: relationalStore.SecurityLevel.S1
6       };
7   }
```

使用关系数据库实现数据持久化，需要通过 getRdbStore 获取一个 RdbStore，示例代码如程序清单 5-6 所示。

程序清单 5-6　Code050402\MyApplication_54\entry\src\main\ets\pages\index.ets

```
//在 pages 中获取 context
    relationalStore.getRdbStore( getContext(this) as common.UIAbilityContext,
    STORE_CONFIG,
      (err: BusinessError, rdbStore: relationalStore.RdbStore) => {
        this.store = rdbStore;
        if (err) {
           console.error('WRYCHH Get RdbStore failed, code is ${err.code},
           message is ${err.message}');
          return;
        }
        console.info('WRYCHH Get RdbStore successfully.');
        this.insert();
      })
```

值得注意的是，如果要在 pages 中获取上下文信息 context，不能直接使用 this.context，而应该使用 getContext(this) as common，这样就可以在 UI 中获取上下文信息了。

获取到 RdbStore 后，定义 insert()函数，调用关系数据库持久化功能的相关接口 insert()接口插入数据，示例代码如程序清单 5-7 所示。

程序清单 5-7　Code050402\MyApplication_54\entry\src\main\ets\pages\index.ets

```
insert() {
  let value1 = "Lisa";
  let value2 = 18;
  let value3 = 100.5;
  let value4 = new Uint8Array([1, 2, 3, 4, 5]);
  const valueBucket: ValuesBucket = {
    'NAME': value1,
    'AGE': value2,
    'SALARY': value3,
     'CODES': value4,
   };

   if (this.store ! = undefined) {
      const SQL_CREATE_TABLE = 'CREATE TABLE IF NOT EXISTS EMPLOYEE (ID
      INTEGER PRIMARY KEY AUTOINCREMENT, NAME TEXT NOT NULL, AGE INTEGER,
      SALARY REAL, CODES BLOB, IDENTITY UNLIMITED INT)';
      //建表 SQL 语句，IDENTITY 为 bigint 类型，SQL 中指定类型为 UNLIMITED INT

     //当数据库创建时，数据库默认版本为 0
     if (this.store.version === 0) {
       this.store.executeSql(SQL_CREATE_TABLE); //创建数据表
       //设置数据库的版本，入参为大于 0 的整数
       this.store.version = 1;
     }

       this.store.insert("EMPLOYEE", valueBucket, (err: BusinessError,
       rowId: number) => {
       if (err) {
           console.error('WRYCHH Insert is failed, code is ${err.code},
           message is ${err.message}');
```