基于 Flutter 开发框架的数据存储访问机制研究*

倪红军，周巧扣
（南京师范大学泰州学院，江苏 泰州 225300）

摘 要：移动端应用程序需要存储与访问的数据越来越多，而移动端设备的存储空间有限，很难满足用户需要，在进行移动端应用程序开发时，选择合适、高效的数据存储与访问机制受到普遍关注。针对 Google 推出的 Flutter 跨平台移动开发框架提供的 key-value 键值对、File 文件、SQLite 数据库和网络 4 种数据存储访问机制，深入阐述了它们的内在工作原理，给出了具体实现方法和实现代码。最后，根据 4 种数据存储访问机制各自的特点和实际的应用程序开发需求，分析了每种机制的应用场景。

关键词：Flutter；数据存储；实现方法；应用场景
中图分类号：TP399 **文献标识码**：A DOI：10.16157/j.issn.0258-7998.212008

中文引用格式：倪红军，周巧扣. 基于 Flutter 开发框架的数据存储访问机制研究[J].电子技术应用，2022，48(8)：107-110，116.
英文引用格式：Ni Hongjun，Zhou Qiaokou. Study of data storage access mechanism based on Flutter development framework[J]. Application of Electronic Technique，2022，48(8)：107-110，116.

Study of data storage access mechanism based on Flutter development framework

Ni Hongjun，Zhou Qiaokou
（Nanjing Normal University Taizhou College，Taizhou 225300，China）

Aabstract：Mobile applications need to store and access more and more data，and the storage space of mobile devices is limited，it is difficult to meet the needs of users. When developing mobile applications，the selection of appropriate and efficient data storage and access mechanism is widely concerned. In view of the four data storage and access mechanisms provided by Google′s Flutter cross-platform mobile development framework，such as key-value pair，File files，SQLite database and network，this paper explains their inner working principles in depth，and gives the specific implementation methods and implementation codes. Finally，according to the characteristics of the four data storage access mechanisms and the actual application development requirements，the application scenarios of each mechanism are analyzed.

Key words：Flutter；data storage；implementation method；application scenarios

0 引言

随着移动互联网技术的快速发展，用户对应用程序的性能、体验等各方面要求都有所增强，例如需要在移动终端平台上做数据缓存来缩短应用程序的响应时间、打开应用程序后能够及时连接网络更新信息保证数据的即时性等。数据缓存主要涉及移动端设备的本地存储和访问的问题，连接网络及时更新信息主要涉及移动端与网络后台数据服务器进行数据交互的问题，在 Flutter 应用开发中，这些问题是由 Flutter 开发框架的存储访问机制来解决的，开发者对该机制的深入理解和合理使用会直接影响应用程序的开发效率和性能，因此对 Flutter 开发框架的数据存储访问机制进行研究具有重要意义。

* 基金项目：江苏高校“教学研究工作”专项课题(2021JSJY081)；2020 年教育部—谷歌产学合作协同育人项目(202002107003)

1 Flutter 简介

Flutter 是谷歌(Google)主导研发并开源的用于创建跨平台的高性能移动应用程序开发框架，开发者可以用一套 Dart 语言代码库高效地构建多平台应用，它全面支持移动、Web、桌面和嵌入式平台[1-2]。自 2018 年 12 月 Flutter1.0 发布后，开发者对 Flutter 的学习和研究不断深入，阿里、蚂蚁金服、腾讯、字节跳动等一线互联网企业在闲鱼、蚂蚁财富、在线教育 App、西瓜视频等重要项目中使用了 Flutter 技术。Flutter 开发框架主要由 Embedder、Flutter Engine 和 Flutter Framework 3 个结构层组成，如图 1 所示。

Embedder(嵌入层)用于将 Flutter 嵌入到各种平台中，其主要任务包括渲染 Surface 设置、线程设置和插件等；Engine（引擎层）是一个用 C++实现的 SDK(Software Development Kit，软件开发包)，其中包括 Skia 引擎(图形

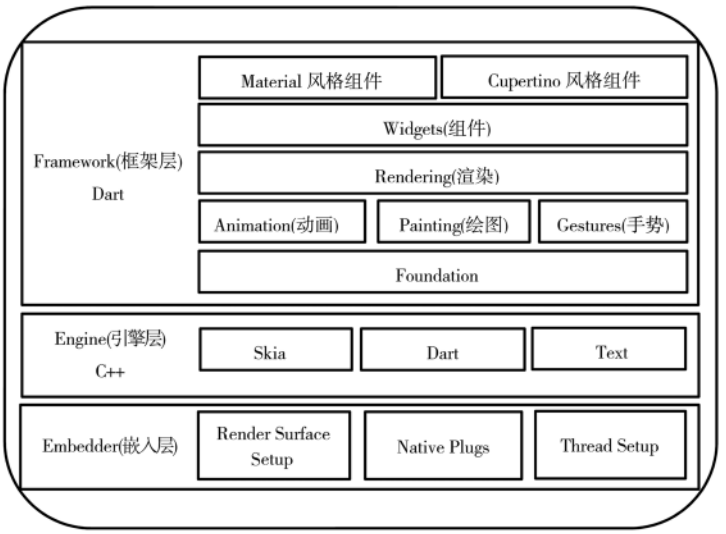

图 1 Flutter 开发框架结构图

绘制)、Dart 运行时和 Text 引擎(文字排版);Framework(框架层)是一个用纯 Dart 语言实现的 SDK,它实现了一套基础库,可以使用 Dart 语言调用 Flutter Engine 的强大能力[3-5]。

2 Flutter 开发框架的存储访问机制

通常在进行应用程序开发时,涉及数据的存储与访问机制有 3 种:本地文件、数据库和网络数据。本地文件和数据库存储与访问机制主要应用于离线应用程序中,例如,登录页面的应用场景中将用户名、密码等信息保存在本地,以便下次登录时不再需要输入这些信息。网络数据的存储与访问机制主要应用于能够及时收集、存储、传输、处理和更新数据的应用程序中,例如,购物页面的应用场景中将用户将选购的商品信息及时反馈给商家,以便商家及时进行后续的操作。基于 Flutter 开发框架的应用程序开发涉及的数据存储与访问也包括文件、数据库和网络数据(云数据)3 类,但从开发者的角度来看,它具体包含 key-value 键值对、File 文件、SQLite 数据库和网络 4 种数据存储访问机制。

2.1 key-value 键值对存储访问机制

2.1.1 原理

key-value 存储与访问机制由 Flutter 社区提供的本地数据存取插件——shared_preferences 实现,其主要存储数据类型包括 Bool、Int、Double、String 和 List 等。该机制其实就是用操作系统平台提供的特定 API 将数据存储到特定文件中,如 Android 平台的 SharedPreferences 和 iOS 平台的 NSUserDefaults。它具有使用方法简单、操作过程异步和数据存储持久化等特点[6]。

2.1.2 实现方法

(1)添加 shared_preferences 依赖

shared_preferences 是 Flutter 提供的以 key-value 格式存储数据的插件,默认开发环境没有安装该插件,需要开发者配置并安装 shared_preferences 插件。

打开 Flutter 项目中的 pubspec.yaml 配置文件,添加 shared_preferences 插件依赖的代码如下:

```
dependencies:
  shared_preferences: ^2.0.5
  flutter:
    sdk: flutter
```

(2)获取 SharedPreferences 对象实例

只有实例化 SharedPreferences 对象后,才能使用 key-value 存储与访问机制读写数据。SharedPreferences 提供的 getInstance( )方法获取 SharedPreferences 对象实例的代码如下:

```
Future<SharedPreferences> preferences=SharedPreferences.getInstance();
```

(3)定义存储 key-value 格式数据的方法

SharedPreferences 提供了 setInt( )、setDouble( )、setBool( )、setString( )和 setStringList( )等方法分别用于存储整型、浮点型、布尔型、字符串型和字符串数组等类型的数据。由于 SharedPreferences 的 getInstance( )方法返回的 SharedPreferences 实例对象为 Future 类型,因此需要通过 SharedPreferences 实例对象异步存储数据。实现代码如下:

```
void saveValue(name, pwd,flag) async {
    SharedPreferences sp=await preferences;
    sp.setString('name', name);                    //保存用户名
    sp.setBool('flag', flag);                      //保存标志
    //保存其他类型数据代码类似
}
```

(4)定义读出 key-value 格式数据的方法

SharedPreferences 提供了 getInt( )、getDouble( )、getBool( )、getString( )和 getStringList( )等方法分别用于读出整型、浮点型、布尔型、字符串型和字符串数组等类型的数据。实现代码如下:

```
void readValue() async {
    SharedPreferences sp=await preferences;
    name=sp.getString('name');          //读出 String 型数据
    flag=sp.getBool('flag');            //读出 Bool 型数据
    //读出其他类型数据代码类似
}
```

SharedPreferences 还提供了用于读出所有的 key 的 getKeys( )方法、判断指定的 key 是否存在的 containsKey( )方法和删除指定的 key 的 remove( )方法。

2.2 File 文件存储访问机制

2.2.1 原理

File 存储访问机制通常应用于将数据以普通文件格式下载或保存到移动终端设备的本地存储空间。在 Flutter 应用程序开发中需要使用 Flutter 社区提供的 path_provider 插件和 Dart 语言的 IO 库来实现。path_provider 插件负责获取基于 Android 或 iOS 平台设备的存储目录;

IO 库负责对存储在相应平台存储目录下的文件进行读写操作。

2.2.2 实现方法

(1)目录操作

在用 Flutter 开发框架进行应用程序开发时,可以用 Dart 的 IO 库提供的文件读写类来实现文件操作,但这些应用程序是运行在文件系统具有差异的 Android 和 iOS 平台上。也就是说,在 Android 和 iOS 平台下应用程序读写文件的文件目录并不一样,所以 Flutter 提供了 path_provider 插件,该插件能够以平台透明的方式,访问基于 Android 和 iOS 平台的移动终端设备文件系统上的临时目录、文档目录和外部存储目录。path_provider 插件中的 getTemporaryDirectory( )方法异步获取临时目录、getApplicationDocumentsDirectory( )方法异步获取文档目录、getExternalStorageDirectory( )方法异步获取外部存储目录[7]。由于这些方法获取目录的操作是异步的,因此需要通过 then( )方法注册将来完成时要调用的回调方法。

在 pubspec.yaml 配置文件中添加 path_provider 插件依赖后,可以用下列代码获取临时目录。

```
String myDir ='';
Future<Directory> tDir = getTemporaryDirectory();
tDir.then((dir) {
    myDir = dir.path;
    print('临时目录为:$myDir');
});
```

Directory 还提供了 create( )、delete( )和 list( )方法在指定目录下创建子目录、删除指定目录和列出指定目录下的内容列表,以便用于不同的应用开发场景。

(2)文件操作

Dart 中 IO 库的 File 类提供的 writeAsString( )方法将 String 类型数据写入文件、writeAsBytes( )方法将 Bytes 类型数据写入文件。写文件包括 read(只读)、write(可读可覆盖)、append(可读可追加)、writeOnly(只写)和 writeOnly-Append(只写追加)等模式。在应用程序文档目录下的 default.ini 文件中,追加写入“我是一段测试文本!”字符串的实现代码如下:

```
String myDir ='';
Future<Directory>tDir=getApplicationDocumentsDirectory();
tDir.then((dir) {
    myDir=dir.path;
    File file=File("$myDir/default.ini");
    file.writeAsString ('我是一段测试文本!', mode:
FileMode.append);
});
```

Dart 中 IO 库的 File 类提供的 readAsString( )方法按 String 类型读出文件内容、readAsBytes( )方法按 Bytes 类型读出文件内容、readAsLines( )方法按 List<String>类型读出文件内容。按 String 类型读出应用程序文档目录下 default.ini 文件的内容的实现代码如下:

```
String myDir ='';
Future<Directory> tDir = getApplicationDocumentsDirectory
();
tDir.then((dir) {
    myDir = dir.path;
    File file = File("$myDir/default.ini");
        file.readAsString().then((info) {
          print(info);          //输出 String 类型文件内容
        });
});
```

2.3 SQLite 数据库存储访问机制

2.3.1 原理

数据库是为了方便对大批量的数据进行增、删、改、查等操作而在应用程序开发中常用的存储访问方式。Flutter 社区提供的 sqflite 插件支持 Android 和 iOS 平台的 SQLite 数据库的存储访问。SQLite 是一款轻量级的关系型数据库,它支持事务和批处理、自动版本管理、标准的 CURD(Create、Update、Retrieve、Delete)操作和在 Android、iOS 系统后台线程中执行数据库操作[8-9],并且只占用很少内存。

基于 Flutter 开发框架开发的应用程序中,SQLite 数据库支持 5 种数据类型:NULL(空值)、INTEGER(整型)、REAL(整型、浮点型)、TEXT(字符串)和 BLOB(Uint8List 型)。虽然 SQLite 数据库并不对字段值进行类型检查,但是在进行数据处理时可能会报类型异常,所以应用程序开发中涉及数据库相关的操作时,要避免存储类型不一致的数据[10-11]。

2.3.2 实现方法

(1)初始化数据库

在 pubspec.yaml 配置文件中添加 sqflite 插件依赖后,用 openDatabase( )方法初始化数据库。例如,打开数据库文件默认路径下的 school.db 文件,如果 school.db文件不存在,则创建该数据库文件,同时创建一个 student 表,该表包含序号(id,INTEGER,主键)、学号(xh,TEXT)、姓名(xm,TEXT)、年龄(age,INTEGER)等字段,实现代码如下:

```
String dbPath ='';
Future<String> path = getDatabasesPath();
path.then((value) async {
    dbPath = value;
    Database db = await openDatabase ('$dbPath/school.
db',                                    //数据库文件
    version:1,                          //版本号
    onCreate: (Database db, int version) async {
        await db.execute(
            'CREATE TABLE student (id INTEGER
PRIMARY KEY,xh TEXT,xm TEXT,age INTEGER)');
```

```
    }, );
});
```

首先通过 getDatabasesPath( )方法异步获取数据库文件默认存放路径,用于指定数据库文件存放路径,并通过 then( )方法注册将来完成时要调用的回调方法;然后在回调方法中用 openDatabase( )方法初始化数据库,如果在 openDatabase( )方法中设定了 version 属性值,则可以通过 onCreate、onUpgrade 和 onDowngrade 的属性值设定的回调方法创建或更新表结构。

(2)操作数据库

Database 类中封装的 rawInsert( )、rawDelete( )、rawUpdate( )、rawQuery( )等方法可以分别实现数据库内容的增、删、改、查等操作。例如,向 student 表中插入一条学号为“09090901”、姓名为“李小红”、年龄为“21”的表记录,实现代码如下:

```
String sql='INSERT INTO student (xh, xm, age) VALUES ("09090901","李小红",21)';
db.rawInsert(sql).then((id) {
    print(id);                                    //输出记录号
});
```

rawDelete( )方法返回删除满足条件记录总数,rawUpdate( )方法返回更新记录总数,rawQuery( )方法返回 List<Map<String, dynamic>>类型的数据,可以使用循环语句遍历所有记录。

2.4 网络数据存储访问机制

2.4.1 原理

网络数据的存储与访问通常由 GET 或 POST 方式的网络请求 API 实现。Flutter 项目中常用的网络请求包括 Dart 语言自带的 HttpClient、原生 HTTP 插件和第三方插件 3 种方式[12]。HttpClient 是实现 HTTP 网络请求的接口类,它的具体操作由 _HttpClient 类实现,_HttpClient 类中封装了 get、post、put、delete、patch 和 head 等请求方法。HTTP 插件支持的 HTTP 请求比较全面,它提供的方法可以方便地访问网络并获取网络资源,不但可以实现 HTML 和 JSON 格式数据的传输,也可以实现文件的上传和下载等功能。实现 HTTP 网络请求的第三方插件有很多,例如国内开发者发布的 dio 插件,因为其具有 Restful API、FormData、拦截器、请求取消、Cookie 管理、文件上传/下载等复杂功能及简单易用的特性受到很多 Flutter 开发者的青睐[13-14]。

2.4.2 实现方法

(1)HttpClient 接口类

HttpClient 接口类包含了一组方法,用于发送 HttpClientRequest 到 HTTP 服务器,并接收 HttpClientResponse 作为服务器的响应,实现代码如下:

```
var httpClient = HttpClient();          //新建 HttpClient 对象
var request = await httpClient.getUrl(Uri.parse(url));
var response = await request.close();
if (response.statusCode == 200) {                    //请求成功
    var responseBody=await response.transform(utf8.decoder).join();
    print(responseBody);                  //输出请求返回的内容
}
```

getUrl( )方法表示以 GET 方式向指定的 URL 地址发起HTTP 请求,close( )方法表示由 HTTP 请求返回 HttpClientResponse 对象,如果请求返回成功,则用.transform(utf8.decoder).join( )方法将返回值转化为字符串;如果返回的字符串是 JSON 格式数据,还需要使用jsonDecode( )方法解析后才能满足开发者需要。

(2)HTTP 插件

在 pubspec.yaml 配置文件中添加 HTTP 依赖后,用“import'package:http/http.dart' as http;”语句导入 http.dart 包。从指定 URL 请求返回内容的实现代码如下:

```
var result = await http.get(url);
if (result.statusCode == 200) {
    print(result.body);                   //输出请求返回的内容
}
```

get( )方法表示以 GET 方式向指定的 URL 地址发起 HTTP 请求,并返回内容。如果向 URL 地址发出的请求带参数,则需要使用如下代码以 POST 方式向指定的URL 地址发起 HTTP 请求。

```
var  result = await http.post(url,body: body);
```

(3)第三方插件——dio

在 pubspec.yaml 配置文件中添加 dio 依赖后,用“import 'package:dio/dio.dart';”语句导入 dio.dart 包。从指定 URL 请求返回内容的实现代码如下:

```
Dio dio = Dio();
var response = await dio.get(url);
if (response.statusCode == 200) {
    print(response.data);                 //输出请求返回的内容
}
```

与 HTTP 插件请求一样,如果向 URL 地址发出的请求带参数,则需要使用如下代码以 POST 方式向指定的URL 地址发起 HTTP 请求。

```
var response = await dio.get(url, queryParameters: pars);
```

3 应用场景分析

基于 Flutter 开发框架的 key-value 键值对、File 文件、SQLite 数据库和网络 4 种数据存储访问机制,从工作原理和实现方法的角度来看各不相同,从存储访问性能和应用场景的角度来看各有其优缺点,从开发者角度来看,应根据实际情况进行选择。下面结合它们的优缺点分析最合适的应用场景。

shared_preferences 插件实现的 key-value 键值对机制,只能用于基本数据类型的存储与访问,所以该方式适用于在移动端设备保存类似用户登录信息、应用程序

(下转第 116 页)

Energy Conversion, 2006, 21(1): 273-284.
[6] KULIGOWSKI R J, BARROS A P. Localized precipitation forecasts from a numerical weather prediction model using artificial neural networks[J]. Weather & Forecasting, 2010, 13(4): 1194-1204.
[7] SOMAN S S, ZAREIPOUR H, MALIK O, et al. A review of wind power and wind speed forecasting methods with different time horizons[C]//North American Power Symposium. IEEE Xplore, 2010: 1-8.
[8] ALEXIADIS M C, DOKOPOULOS P S, SAHSAMANOGLOU H S. Wind speed and power forecasting based on spatial correlation models[J]. IEEE Transactions on Energy Conversion, 1999, 14(3): 836-842.
[9] HAMEL P, ECK D. Learning features from music audio with deep belief networks[C]//International Society for Music Information Retrieval Conference, Ismir 2010, Utrecht, Netherlands, August. DBLP, 2010: 339-344.
[10] 胡昭华, 宋耀良. 基于 Autoencoder 网络的数据降维和重构[J]. 电子与信息学报, 2009, 31(5): 1189-1192.
[11] GROVER A, KAPOOR A, HORVITZ E. A deep hybrid model for weather forecasting[C]//ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015: 379-386.
[12] 国家电网公司. Q/GDW 588—2011 风电功率预测功能规范[M]. 北京: 中国电力出版社, 2011.
[13] HINTON G E. Deep belief networks[J]. Scholarpedia, 2009, 4(6): 5947.
[14] HINTON G E, OSINDERO S, TEH Y, et al. A fast learning algorithm for deep belief nets[J]. Neural Computation, 2006, 18(7): 1527-1554.
[15] BERGSTRA J, BENGIO Y. Random search for hyper-para-

(下转第 122 页)

(上接第 110 页)

配置信息等数据格式简单、数据量小及需要缩短应用程序响应时间的场景。而对于数据量比较大、数据格式比较复杂的应用场景，适合选择 File 文件和 SQLite 数据库存储与访问机制。如果经常需要大批量处理数据，则选择 SQLite 数据库更合适。在需要实时更新移动端设备应用程序数据的场景，或者需要将移动端设备的数据发送给远端服务器进行处理的场景，则需要选择网络存储与访问机制，随着网络性能和移动终端设备处理能力的提升，现在越来越多的应用程序选择这种机制。

4 结论

随着 5G 技术的不断发展和成熟，5G 在各领域的应用速度也不断加快，应用程度也日益深化，同时也为移动应用的开发者们提供了更大的发挥空间。但是移动应用程序的开发效率和性能已经成为当前移动应用开发最为关注的问题，而数据的存储与访问是开发应用程序时需要解决的最基本的问题[15]。本文对 Flutter 开发框架下 4 种存储与访问机制的实现原理、实现方法进行研究与分析，并展示出具体的实现代码，为 Flutter 项目开发者提供了有效范例。最后，通过对 key-value 键值对、File 文件、SQLite 数据库和网络 4 种数据存储访问机制优缺点的分析，给出了应用场景建议，为 Flutter 项目开发者选择合适的数据存储与访问机制提供了重要依据。

参考文献

[1] Frank Zammetti. Flutter 实战[M]. 贡国栋, 任强, 译. 北京: 清华大学出版社, 2020.
[2] CHRIS S. Dart 语言程序设计[M]. 韩国恺, 译. 北京: 人民邮电出版社, 2012.
[3] Alessandro Biessek. Flutter 入门与实践[M]. 李强, 译. 北京: 清华大学出版社, 2020.
[4] 亢少军. Dart 语言实战: 基于 Flutter 框架的程序开发[M]. 北京: 清华大学出版社, 2020.
[5] TRAN T. Flutter native performance and expressive UI/UX[D]. Helsinki: Metropolia University of Applied Sciences, 2020.
[6] Flutter Favorite. Shared_preference[EB/OL]. (2021-04-16)[2021-08-05]. https://pub.flutter-io.cn/packages/shared_preferences.
[7] Flutter Favorite. Path_provide[EB/OL]. (2021-04-17)[2021-08-05]. https://pub.flutter-io.cn/packages/path_provider.
[8] Flutter Favorite. Sqflite[EB/OL]. (2021-04-16)[2021-08-05]. https://pub.flutter-io.cn/packages/sqflite.
[9] 倪红军. Flutter 开发零基础入门[M]. 北京: 清华大学出版社, 2021.
[10] 何瑞群. Flutter 从 0 到 1 构建大前端应用[M]. 北京: 电子工业出版社, 2019.
[11] 张益珲. 用 Flutter 极速构建原生应用[M]. 北京: 清华大学出版社, 2019.
[12] 汪若彪, 史国芳, 施尹志, 等. 基于 Flutter 的跨平台的工业移动应用软件设计[J]. 仪表技术, 2021(2): 1-3, 70.
[13] 杜文. Flutter 实战[M]. 北京: 机械工业出版社, 2020.
[14] 萧文翰. Flutter 从 0 基础到 App 上线[M]. 北京: 电子工业出版社, 2020.
[15] 黄阳. 一个跨平台的移动应用开发和运行支撑平台的研究与实现[D]. 南京: 南京大学, 2013.

(收稿日期: 2021-08-05)

作者简介:
倪红军(1975-), 男, 硕士, 副教授, 主要研究方向: 移动应用开发技术、教学改革研究、实验室建设与管理。
周巧扣(1982-), 男, 硕士, 副教授, 主要研究方向: 管理信息系统、机器学习。

版权声明