

## 25 | 本地存储与数据库的使用和优化

2019-08-24 陈航

Flutter核心技术与实战

[进入课程 >](#)



讲述：陈航

时长 11:16 大小 10.33M



你好，我是陈航。

在上一篇文章中，我带你一起学习了 Flutter 的网络编程，即如何建立与 Web 服务器的通信连接，以实现数据交换，以及如何解析结构化后的通信信息。

其中，建立通信连接在 Flutter 中有三种基本方案，包括 HttpClient、http 与 dio。考虑到 HttpClient 与 http 并不支持复杂的网络请求行为，因此我重点介绍了如何使用 dio 实现资源访问、接口数据请求与提交、上传及下载文件、网络拦截等高级操作。

而关于如何解析信息，由于 Flutter 并不支持反射，因此只提供了手动解析 JSON 的方式：把 JSON 转换成字典，然后给自定义的类属性赋值即可。

正因为有了网络，我们的 App 拥有了与外界进行信息交换的通道，也因此具备了更新数据的能力。不过，经过交换后的数据通常都保存在内存中，而应用一旦运行结束，内存就会被释放，这些数据也就随之消失了。

因此，我们需要把这些更新后的数据以一定的形式，通过一定的载体保存起来，这样应用下次运行时，就可以把数据从存储的载体中读出来，也就实现了**数据的持久化**。

数据持久化的应用场景有很多。比如，用户的账号登录信息需要保存，用于每次与 Web 服务验证身份；又比如，下载后的图片需要缓存，避免每次都要重新加载，浪费用户流量。

由于 Flutter 仅接管了渲染层，真正涉及到存储等操作系统底层行为时，还需要依托于原生 Android、iOS，因此与原生开发类似的，根据需要持久化数据的大小和方式不同，Flutter 提供了三种数据持久化方法，即文件、SharedPreferences 与数据库。接下来，我将与你详细讲述这三种方式。

## 文件

文件是存储在某种介质（比如磁盘）上指定路径的、具有文件名的一组有序信息的集合。从其定义看，要想以文件的方式实现数据持久化，我们首先需要确定一件事儿：数据放在哪儿？这，就意味着要定义文件的存储路径。

Flutter 提供了两种文件存储的目录，即**临时（Temporary）目录与文档（Documents）目录**：


临时目录是操作系统可以随时清除的目录，通常被用来存放一些不重要的临时缓存数据。这个目录在 iOS 上对应着 `NSTemporaryDirectory` 返回的值，而在 Android 上则对应着 `getCacheDir` 返回的值。

文档目录则是只有在删除应用程序时才会被清除的目录，通常被用来存放应用产生的重要数据文件。在 iOS 上，这个目录对应着 `NSDocumentDirectory`，而在 Android 上则对应着 `AppData` 目录。

接下来，我通过一个例子与你演示如何在 Flutter 中实现文件读写。


在下面的代码中，我分别声明了三个函数，即创建文件目录函数、写文件函数与读文件函数。这里需要注意的是，由于文件读写是非常耗时的操作，所以这些操作都需要在异步环境

下进行。另外，为了防止文件读取过程中出现异常，我们也需要在外层包上 try-catch：

 复制代码

```
1 // 创建文件目录
2 Future<File> get _localFile async {
3   final directory = await getApplicationDocumentsDirectory();
4   final path = directory.path;
5   return File('$path/content.txt');
6 }
7 // 将字符串写入文件
8 Future<File> writeContent(String content) async {
9   final file = await _localFile;
10  return file.writeAsString(content);
11 }
12 // 从文件读出字符串
13 Future<String> readContent() async {
14   try {
15     final file = await _localFile;
16     String contents = await file.readAsString();
17     return contents;
18   } catch (e) {
19     return "";
20   }
21 }
```

有了文件读写函数，我们就可以在代码中对 content.txt 这个文件进行读写操作了。在下面的代码中，我们往这个文件写入了一段字符串后，隔了一会又把它读了出来：

 复制代码

```
1 writeContent("Hello World!");
2 ...
3 readContent().then((value)=>print(value));
```

除了字符串读写之外，Flutter 还提供了二进制流的读写能力，可以支持图片、压缩包等二进制文件的读写。这些内容不是本次分享的重点，如果你想要深入研究的话，可以查阅[官方文档](#)。


## SharedPreferences

文件比较适合大量的、有序的数据持久化，如果我们只是需要缓存少量的键值对信息（比如记录用户是否阅读了公告，或是简单的计数），则可以使用 SharedPreferences。

SharedPreferences 会以原生平台相关的机制，为简单的键值对数据提供持久化存储，即在 iOS 上使用 NSUserDefaults，在 Android 使用 SharedPreferences。


接下来，我通过一个例子来演示在 Flutter 中如何通过 SharedPreferences 实现数据的读写。在下面的代码中，我们将计数器持久化到了 SharedPreferences 中，并为它分别提供了读方法和递增写入的方法。

这里需要注意的是，setter（setInt）方法会同步更新内存中的键值对，然后将数据保存至磁盘，因此我们无需再调用更新方法强制刷新缓存。同样地，由于涉及到耗时的文件读写，因此我们必须以异步的方式对这些操作进行包装：

 复制代码

```
1 // 读取 SharedPreferences 中 key 为 counter 的值
2 Future<int>_loadCounter() async {
3   SharedPreferences prefs = await SharedPreferences.getInstance();
4   int counter = (prefs.getInt('counter') ?? 0);
5   return counter;
6 }
7
8 // 递增写入 SharedPreferences 中 key 为 counter 的值
9 Future<void>_incrementCounter() async {
10  SharedPreferences prefs = await SharedPreferences.getInstance();
11  int counter = (prefs.getInt('counter') ?? 0) + 1;
12  prefs.setInt('counter', counter);
13 }
```

在完成了计数器存取方法的封装后，我们就可以在代码中随时更新并持久化计数器数据了。在下面的代码中，我们先是读取并打印了计数器数据，随后将其递增，并再次把它读取打印：

 复制代码

```
1 // 读出 counter 数据并打印
2 _loadCounter().then((value)=>print("before:$value"));
3
4 // 递增 counter 数据后，再次读出并打印
5 _incrementCounter().then((_) {
```

```
6  _loadCounter().then((value)=>print("after:$value"));
7  });
```

可以看到，SharedPreferences 的使用方式非常简单方便。不过需要注意的是，以键值对的方式只能存储基本类型的数据，比如 int、double、bool 和 string。


## 数据库

SharedPreferences 的使用固然方便，但这种方式只适用于持久化少量数据的场景，我们并不能用它来存储大量数据，比如文件内容（文件路径是可以的）。

如果我们需要持久化大量格式化后的数据，并且这些数据还会以较高的频率更新，为了考虑进一步的扩展性，我们通常会选用 sqlite 数据库来应对这样的场景。与文件和 SharedPreferences 相比，数据库在数据读写上可以提供更快、更灵活的解决方案。

接下来，我就以一个例子分别与你介绍数据库的使用方法。

我们以上一篇文章中提到的 Student 类为例：


 复制代码

```
1  class Student{
2    String id;
3    String name;
4    int score;
5    // 构造方法
6    Student({this.id, this.name, this.score,});
7    // 用于将 JSON 字典转换成类对象的工厂类方法
8    factory Student.fromJson(Map<String, dynamic> parsedJson){
9      return Student(
10        id: parsedJson['id'],
11        name : parsedJson['name'],
12        score : parsedJson ['score'],
13      );
14    }
15  }
```

JSON 类拥有一个可以将 JSON 字典转换成类对象的工厂类方法，我们也可以提供将类对象反过来转换成 JSON 字典的实例方法。因为最终存入数据库的并不是实体类对象，而是




字符串、整型等基本类型组成的字典，所以我们可以通过这两个方法，实现数据库的读写。同时，我们还分别定义了 3 个 Student 对象，用于后续插入数据库：

 复制代码

```
1 class Student{
2     ...
3     // 将类对象转换成 JSON 字典，方便插入数据库
4     Map<String, dynamic> toJson() {
5         return {'id': id, 'name': name, 'score': score,};
6     }
7 }
8
9 var student1 = Student(id: '123', name: '张三', score: 90);
10 var student2 = Student(id: '456', name: '李四', score: 80);
11 var student3 = Student(id: '789', name: '王五', score: 85);
```

有了实体类作为数据库存储的对象，接下来就需要创建数据库了。在下面的代码中，我们通过 openDatabase 函数，给定了一个数据库存储地址，并通过数据库表初始化语句，创建了一个用于存放 Student 对象的 students 表：

 复制代码

```
1 final Future<Database> database = openDatabase(
2     join(await getDatabasesPath(), 'students_database.db'),
3     onCreate: (db, version)=>db.execute("CREATE TABLE students(id TEXT PRIMARY KEY, name '
4     version: 1,
5 );
```


以上代码属于通用的数据库创建模板，有两个地方需要注意：

1. 在设定数据库存储地址时，使用 join 方法对两段地址进行拼接。join 方法在拼接时会使用操作系统的路径分隔符，这样我们就无需关心路径分隔符究竟是 “/” 还是 “\” 了。
2. 在创建数据库时，传入了一个参数 version:1，在 onCreate 方法的回调里面也有一个参数 version。前者代表当前版本的数据库版本，后者代表用户手机上的数据库版本。

比如，我们的应用有 1.0、1.1 和 1.2 三个版本，在 1.1 把数据库 version 升级到了 2。考虑到用户的升级顺序并不总是连续的，可能会直接从 1.0 升级到 1.2。因此我们可以在


onCreate 函数中，根据数据库当前版本和用户手机上的数据库版本进行比较，制定数据库升级方案。

数据库创建好了之后，接下来我们就可以把之前创建的 3 个 Student 对象插入到数据库中了。数据库的插入需要调用 insert 方法，在下面的代码中，我们将 Student 对象转换成了 JSON，在指定了插入冲突策略（如果同样的对象被插入两次，则后者替换前者）和目标数据库表后，完成了 Student 对象的插入：

 复制代码

```
1 Future<void> insertStudent(Student std) async {
2     final Database db = await database;
3     await db.insert(
4         'students',
5         std.toJson(),
6         // 插入冲突策略，新的替换旧的
7         conflictAlgorithm: ConflictAlgorithm.replace,
8     );
9 }
10 // 插入 3 个 Student 对象
11 await insertStudent(student1);
12 await insertStudent(student2);
13 await insertStudent(student3);
```

数据完成插入之后，接下来我们就可以调用 query 方法把它们取出来了。需要注意的是，写入的时候我们是一个接一个地有序插入，读的时候我们则采用批量读的方式（当然也可以指定查询规则读特定对象）。读出来的数据是一个 JSON 字典数组，因此我们还需要把它转换成 Student 数组：

 复制代码

```
1 Future<List<Student>> students() async {
2     final Database db = await database;
3     final List<Map<String, dynamic>> maps = await db.query('students');
4     return List.generate(maps.length, (i)=>Student.fromJson(maps[i]));
5 }
6
7 // 读取数据库中的 Student 对象集合
8 students().then((list)=>list.forEach((s)=>print(s.name)));
```

可以看到，在面对大量格式化的数据模型读取时，数据库提供了更快、更灵活的持久化解决方案。

除了基础的数据库读写操作之外，sqlite 还提供了更新、删除以及事务等高级特性，这与原生 Android、iOS 上的 SQLite 或是 MySQL 并无不同，因此这里就不再赘述了。你可以参考 sqflite 插件的[API 文档](#)，或是查阅[SQLite 教程](#)了解具体的使用方法。

## 总结

好了，今天的分享就这里。我们简单回顾下今天学习的内容吧。

首先，我带你学习了文件，这种最常见的数据持久化方式。Flutter 提供了两类目录，即临时目录与文档目录。我们可以根据实际需求，通过写入字符串或二进制流，实现数据的持久化。

然后，我通过一个小例子和你讲述了 SharedPreferences，这种适用于持久化小型键值对的存储方案。

最后，我们一起学习了数据库。围绕如何将一个对象持久化到数据库，我与你介绍了数据库的创建、写入和读取方法。可以看到，使用数据库的方式虽然前期准备工作多了不少，但面对持续变更的需求，适配能力和灵活性都更强了。

数据持久化是 CPU 密集型运算，因此数据存取均会大量涉及到异步操作，所以请务必使用异步等待或注册 then 回调，正确处理读写操作的时序关系。

我把今天分享所涉及到的知识点打包到了[GitHub](#)中，你可以下载下来，反复运行几次，加深理解与记忆。


## 思考题

最后，我给你留下两道思考题吧。

1. 请你分别介绍一下文件、SharedPreferences 和数据库，这三种持久化数据存储方式的适用场景。
2. 我们的应用经历了 1.0、1.1 和 1.2 三个版本。其中，1.0 版本新建了数据库并创建了 Student 表，1.1 版本将 Student 表增加了一个字段 age ( ALTER TABLE students

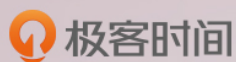


ADD age INTEGER )。请你写出 1.1 版本及 1.2 版本的数据库创建代码。

 复制代码

```
1 //1.0 版本数据库创建代码
2 final Future<Database> database = openDatabase(
3   join(await getDatabasesPath(), 'students_database.db'),
4   onCreate: (db, version)=>db.execute("CREATE TABLE students(id TEXT PRIMARY KEY, name TEXT,
5     version: 1,
6   );
```

欢迎你在评论区给我留言分享你的观点，我会在下一篇文章中等待你！感谢你的收听，也欢迎你把这篇文章分享给更多的朋友一起阅读。



## Flutter 核心技术与实战

来自 Google 的高性能跨平台开发框架

陈航

美团点评高级技术专家



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 24 | HTTP网络编程与JSON解析

### 精选留言 (2)



 写留言



许章童  
2019-08-24

思考题:

1.文件用来存储图片，视频类的大文件，SharedPreferences 用来存储一些键值对，比如记住用户名，数据库用来存储类似表格有关系的数据行。

2.

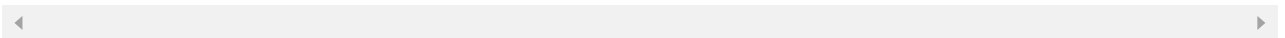
//1.1 版本数据库创建代码...

展开 ▾

作者回复: 数据库创建的语句不对：

1.1.0，1.1和1.2传入的version完全一样，没办法区分数据库版本

2.数据库升级的代码不太适合用if-else判断具体版本去写适配策略，版本一多适配代码就乱了。建议用none break的switch-case去写



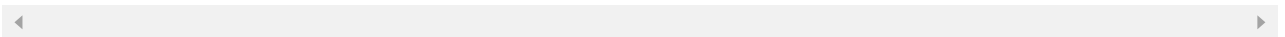
1



汪帅  
2019-08-24

我想请问一下关于获取系统信息怎么做啊？例如通讯录，安装的APP信息等等

作者回复: 需要在原生代码宿主写方法通道来实现了。具体可以参考26节的内容



2

