存储选项

存储快览

- 将共享首选项用于原始数据
- 将内部设备存储用于私有数据
- 将外部存储用于非私有大型数据集
- 将 SQLite 数据库用于结构化存储

本文内容

另请参阅

- 内容提供程序和内容解析程序

Android 为您提供了多种选项来保存永久性应用数据。您所选择的解决方案取决于您的特定需求，例如数据应该是应用的私有数据，还是可供其他应用（和用户）访问，以及您的数据需要多少空间等。

您的数据存储选项如下：

共享首选项

在键值对中存储私有原始数据。

内部存储

在设备内存中存储私有数据。

外部存储

在共享的外部存储中存储公共数据。

SQLite 数据库

在私有数据库中存储结构化数据。

网络连接

在网络中使用您自己的网络服务器存储数据。

Android 为您提供了一种方法 — 使用内容提供程序将您的数据（甚至是您的私有数据）公开给其他应用。 内容提供程序是一个可选组件，可根据您希望施加的任何限制公开您的应用数据的读/写访问权限。 如需了解有关使用内容提供程序的更多信息，请参阅内容提供程序文档。

使用共享首选项

`SharedPreferences` 类提供了一个通用框架，以便您能够保存和检索原始数据类型的永久性键值对。 您可以使用 `SharedPreferences` 来保存任何原始数据：布尔值、浮点值、整型值、长整型和字符串。 此数据将跨多个用户会话永久保留（即使您的应用已终止亦如此）。

要获取应用的 `SharedPreferences` 对象，请使用以下两个方法之一：

- `getSharedPreferences()` - 如果您需要多个按名称（使用第一个参数指定）识别的首选项文件，请使用此方法。
- `getPreferences()` - 如果您只需要一个用于 Activity 的首选项文件，请使用此方法。 由于这将是用于 Activity 的唯一首选项文件，因此无需提供名称。

> **用户首选项**
>
> 严格来说，共享首选项并非用于保存“用户首选项”，例如用户所选择的铃声。 如果您有兴趣为您的应用创建用户首选项，请参阅 `PreferenceActivity`，其中为您提供了一个 Activity 框架，用于创建将会自动永久保留（通过共享首选项）的用户首选项。

要写入值：

1. 调用 `edit()` 以获取 `SharedPreferences.Editor`。
2. 使用 `putBoolean()` 和 `putString()` 等方法添加值。
3. 使用 `commit()` 提交新值

要读取值，请使用 `getBoolean()` 和 `getString()` 等 `SharedPreferences` 方法。

以下是在计算器中保存静音按键模式首选项的示例：

```
public class Calc extends Activity {
    public static final String PREFS_NAME = "MyPrefsFile";

    @Override
    protected void onCreate(Bundle state){
       super.onCreate(state);
       . . .

       // Restore preferences
       SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
       boolean silent = settings.getBoolean("silentMode", false);
       setSilent(silent);
    }

    @Override
    protected void onStop(){
       super.onStop();

      // We need an Editor object to make preference changes.
      // All objects are from android.context.Context
      SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
      SharedPreferences.Editor editor = settings.edit();
      editor.putBoolean("silentMode", mSilentMode);

      // Commit the edits!
      editor.commit();
    }
}
```

使用内部存储

您可以直接在设备的内部存储中保存文件。默认情况下，保存到内部存储的文件是应用的私有文件，其他应用（和用户）不能访问这些文件。当用户卸载您的应用时，这些文件也会被移除。

要创建私有文件并写入到内部存储：

1. 使用文件名称和操作模式调用 `openFileOutput()`。 这将返回一个 `FileOutputStream`。
2. 使用 `write()` 写入到文件。
3. 使用 `close()` 关闭流式传输。

例如：

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

`MODE_PRIVATE` 将会创建文件（或替换具有相同名称的文件），并将其设为应用的私有文件。 其他可用模式包括：`MODE_APPEND`、`MODE_WORLD_READABLE` 和 `MODE_WORLD_WRITEABLE`。

> **注**：自 API 级别 17 以来，常量 `MODE_WORLD_READABLE` 和 `MODE_WORLD_WRITEABLE` 已被弃用。从 Android N 开始，使用这些常量将会导致引发 `SecurityException`。这意味着，面向 Android N 和更高版本的应用无法按名称共享私有文件，尝试共享“file://”URI 将会导致引发 `FileUriExposedException`。 如果您的应用需要与其他应用共享私有文件，则可以将 `FileProvider` 与 `FLAG_GRANT_READ_URI_PERMISSION` 配合使用。另请参阅**共享文件**。

要从内部存储读取文件：

1. 调用 `openFileInput()` 并向其传递要读取的文件名称。 这将返回一个 `FileInputStream`。
2. 使用 `read()` 读取文件字节。
3. 然后使用 `close()` 关闭流式传输。

> **提示**：如果在编译时想要保存应用中的静态文件，请在项目的 `res/raw/` 目录中保存该文件。 可以使用 `openRawResource()` 打开该资源并传递 `R.raw.<filename>` 资源 ID。 此方法将返回一个 `InputStream`，您可以使用该流式传输读取文件（但不能写入到原始文件）。

保存缓存文件

如果您想要缓存一些数据，而不是永久存储这些数据，应该使用 `getCacheDir()` 来打开一个 `File`，它表示您的应用应该将临时缓存文件保存到的内部目录。

当设备的内部存储空间不足时，Android 可能会删除这些缓存文件以回收空间。 但您不应该依赖系统来为您清理这些文件， 而应该始终自行维护缓存文件，使其占用的空间保持在合理的限制范围内（例如 1 MB）。 当用户卸载您的应用时，这些文件也会被移除。

其他实用方法

`getFilesDir()`

获取在其中存储内部文件的文件系统目录的绝对路径。

`getDir()`

在您的内部存储空间内创建（或打开现有的）目录。

`deleteFile()`

删除保存在内部存储的文件。

`fileList()`

返回您的应用当前保存的一系列文件。

使用外部存储

每个兼容 Android 的设备都支持可用于保存文件的共享“外部存储”。 该存储可能是可移除的存储介质（例如 SD 卡）或内部（不可移除）存储。 保存到外部存储的文件是全局可读取文件，而且，在计算机上启用 USB 大容量存储以传输文件后，可由用户修改这些文件。

> **注意**：如果用户在计算机上装载了外部存储或移除了介质，则外部存储可能变为不可用状态，并且在您保存到外部存储的文件上没有实施任何安全性。 所有应用都能读取和写入放置在外部存储上的文件，并且用户可以移除这些文件。

使用作用域目录访问

在 Android 7.0 或更高版本中，如果您需要访问外部存储上的特定目录，请使用作用域目录访问。 作用域目录访问可简化您的应用访问标准外部存储目录（例如 `Pictures` 目录）的方式，并提供简单的权限 UI，清楚地详细介绍应用正在请求访问的目录。 有关作用域目录访问的更多详情，请参阅使用作用域目录访问。

获取外部存储的访问权限

要读取或写入外部存储上的文件，您的应用必须获取 `READ_EXTERNAL_STORAGE` 或 `WRITE_EXTERNAL_STORAGE` 系统权限。 例如：

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

如果您同时需要读取和写入文件，则只需请求 `WRITE_EXTERNAL_STORAGE` 权限，因为此权限也隐含了读取权限要求。

> **注**：从 Android 4.4 开始，如果您仅仅读取或写入应用的私有文件，则不需要这些权限。 如需了解更多信息，请参阅下面有关保存应用私有文件的部分。

检查介质可用性

在使用外部存储执行任何工作之前，应始终调用 `getExternalStorageState()` 以检查介质是否可用。介质可能已装载到计算机，处于缺失、只读或其他某种状态。 例如，以下是可用于检查可用性的几种方法：

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

`getExternalStorageState()` 方法将返回您可能需要检查的其他状态（例如介质是否处于共享 [连接到计算]、完全缺失、错误移除等状态）。 当您的应用需要访问介质时，您可以使用这些状态向用户通知更多信息。

保存可与其他应用共享的文件

一般而言，应该将用户可通过您的应用获取的新文件保存到设备上的“公共”位置，以便其他应用能够在其中访问这些文件，并且用户也能轻松地从该设备复制这些文件。 执行此操作时，应使用共享的公共目录之一，例如 `Music/`、`Pictures/` 和 `Ringtones/` 等。

要获取表示相应的公共目录的 `File`，请调用 `getExternalStoragePublicDirectory()`，向其传递您需要的目录类型，例如 `DIRECTORY_MUSIC`、`DIRECTORY_PICTURES`、`DIRECTORY_RINGTONES` 或其他类型。通过将您的文件保存到相应的媒体类型目录，系统的媒体扫描程序可以在系统中正确地归类您的文件（例如铃声在系统设置中显示为铃声而不是音乐）。

例如，以下方法在公共图片目录中创建了一个用于新相册的目录：

> **在媒体扫描程序中隐藏您的文件**
>
> 在您的外部文件目录中包含名为 `.nomedia` 的空文件（注意文件名中的点前缀）。 这将阻止媒体扫描程序读取您的媒体文件，并通过 `MediaStore` 内容提供程序将其提供给其他应用。 但如果您的文件真正是应用的私有文件，则应该将其保存在应用私有的目录中。

```
public File getAlbumStorageDir(String albumName) {
    // Get the directory for the user's public pictures directory.
    File file = new File(Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

保存应用私有文件

如果您正在处理的文件不适合其他应用使用（例如仅供您的应用使用的图形纹理或音效），则应该通过调用 `getExternalFilesDir()` 来使用外部存储上的私有存储目录。此方法还会采用 `type` 参数指定子目录的类型（例如 `DIRECTORY_MOVIES`）。 如果您不需要特定的媒体目录，请传递 `null` 以接收应用私有目录的根目录。

从 Android 4.4 开始，读取或写入应用私有目录中的文件不再需要 `READ_EXTERNAL_STORAGE` 或 `WRITE_EXTERNAL_STORAGE` 权限。 因此，您可以通过添加 `maxSdkVersion` 属性来声明，只能在较低版本的 Android 中请求该权限：

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
                     android:maxSdkVersion="18" />
    ...
</manifest>
```

> **注**：当用户卸载您的应用时，此目录及其内容将被删除。此外，系统媒体扫描程序不会读取这些目录中的文件，因此不能从 `MediaStore` 内容提供程序访问这些文件。 同样，**不应将这些目录**用于最终属于用户的媒体，例如使用您的应用拍摄或编辑的照片或用户使用您的应用购买的音乐等 — 这些文件应保存在公共目录中。

有时，已分配某个内部存储器分区用作外部存储的设备可能还提供了 SD 卡槽。在使用运行 Android 4.3 和更低版本的这类设备时，`getExternalFilesDir()` 方法将仅提供内部分区的访问权限，而您的应用无法读取或写入 SD 卡。不过，从 Android 4.4 开始，可通过调用 `getExternalFilesDirs()` 来同时访问两个位置，该方法将会返回包含各个位置条目的 `File` 数组。 数组中的第一个条目被视为外部主存储；除非该位置已满或不可用，否则应该使用该位置。 如果您希望在支持 Android 4.3 和更低版本的同时访问两个可能的位置，请使用支持库中的静态方法 `ContextCompat.getExternalFilesDirs()`。 在 Android 4.3 和更低版本中，此方法也会返回一个 `File` 数组，但其中始终仅包含一个条目。

> **注意** 尽管 `MediaStore` 内容提供程序不能访问 `getExternalFilesDir()` 和 `getExternalFilesDirs()` 所提供的目录，但其他具有 `READ_EXTERNAL_STORAGE` 权限的应用仍可访问外部存储上的所有文件，包括上述文件。 如果您需要完全限制您的文件的访问权限，则应该转而将您的文件写入到内部存储。

保存缓存文件

要打开表示应该将缓存文件保存到的外部存储目录的 `File`，请调用 `getExternalCacheDir()`。 如果用户卸载您的应用，这些文件也会被自动删除。

与前述 `ContextCompat.getExternalFilesDirs()` 相似，您也可以通过调用 `ContextCompat.getExternalCacheDirs()` 来访问辅助外部存储（如果可用）上的缓存目录。

> **提示**：为节省文件空间并保持应用性能，您应该在应用的整个生命周期内仔细管理您的缓存文件并移除其中不再需要的文件，这一点非常重要。

使用数据库

Android 提供了对 SQLite 数据库的完全支持。应用中的任何类（不包括应用外部的类）均可按名称访问您所创建的任何数据库。

创建新 SQLite 数据库的推荐方法是创建 `SQLiteOpenHelper` 的子类并覆盖 `onCreate()` 方法，在此方法中，您可以执行 SQLite 命令以创建数据库中的表。 例如：

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

然后您可以使用已定义的构造函数获取 `SQLiteOpenHelper` 实现的实例。 要从数据库执行写入和读取操作，请分别调用 `getWritableDatabase()` 和 `getReadableDatabase()`。二者都会返回一个表示数据库的 `SQLiteDatabase` 对象，并提供用于 SQLite 操作的方法。

您可以使用 `SQLiteDatabase query()` 方法来执行 SQLite 查询，这些方法可接受各种查询参数，例如要查询的表、投影、选择、列、分组和其他参数。 对于复杂的查询，例如需要列别名的查询，应该使用 `SQLiteQueryBuilder`，它将提供多种便捷的方法来构建查询。

每个 SQLite 查询都会返回一个指向该查询找到的所有行的 `Cursor`。 您始终可以使用 `Cursor` 机制来浏览数据库查询结果，以及读取行和列。

如需演示 Android 中的 SQLite 数据库使用方法的示例应用，请参阅 **记事本**和 **可搜索字典**应用。

> Android 没有实施标准 SQLite 概念之外的任何限制。我们推荐包含一个可用作唯一 ID 的自动增量值关键字段，以便快速查找记录。 私有数据不要求这样做，但如果您实现了一个**内容提供程序**，则必须包含使用 `BaseColumns._ID` 常量的唯一 ID。

数据库调试

Android SDK 包含一项 `sqlite3` 数据库工具，利用此工具可以浏览表内容，运行 SQL 命令，以及在 SQLite 数据库上执行其他实用功能。 请参阅**从远程 shell 检查 sqlite3 数据库**，了解如何运行此工具。

使用网络连接

您可以使用网络（如果可用）来存储和检索有关您自己的网络服务的数据。 要执行网络操作，请使用以下包中的类：

- `java.net.*`
- `android.net.*`