|  |  |
| --- | --- |
| **Работа с файловой системой сохранение файлов** |  |

|  |
| --- |
| Добавил(а) microsin |
| **Android** использует файловую систему, которая аналогична дисковым файловым системам на других платформах. Эта лекция описывает, как работать с файловой системой Android для чтения и записи файлов с помощью **File API** (перевод документации [1]).  Объект [File](http://developer.android.com/reference/java/io/File.html) подходит для чтения или записи больших объемов данных в порядке от начала до конца, без пропусков. Например, это хорошо подходит для файлов картинок или для различных обменов данными через сеть. Здесь будет показано, как выполнять базовые файловые операции в Вашем приложении. Подразумевается, что Вы знакомы с файловой системой Linux и стандартной системой ввода/вывода файлов в (standard file input/output API) в [java.io](http://developer.android.com/reference/java/net/URL.html).  [**Выбор между внутренним и внешним хранилищем (Internal Storage, External Storage)**]  Все устройства Android имеют две области хранения файлов (file storage area): "*internal*" и "*external*" storage. Эти имена появились на первых стадиях развития Android, когда большинство устройств поставлялись со встроенной энергонезависимой памятью FLASH (**internal storage**) плюс извлекаемый носитель памяти, такой как **micro SD card**(**external storage**). Некоторые устройства делили постоянную область хранения (permanent storage space) на разделы "internal" и "external", так что даже без наличия внешнего извлекаемого хранилища (removable storage medium) всегда имеется 2 пространства хранения, и поведение API всегда одинаково - независимо от того, есть в наличии внешнее хранилище или нет. Следующие списки подводят общую черту под фактическими различиями каждого пространства хранения.   |  |  | | --- | --- | | **Internal storage** (внутреннее, неизвлекаемое хранилище): | **External storage** (внешнее, извлекаемое хранилище): | | • Оно всегда доступно. • Файлы, которые сохранены здесь, по умолчанию доступны только Вашего приложения. • Не требуется запрашивать разрешения на доступ к internal storage для Вашего приложения. • Когда пользователь деинсталлирует Ваше приложение, то система также удалит все файлы приложения с internal storage.  Вывод: internal storage лучше подходит, когда ни другие пользователи, ни другие приложения не могли получить доступа к Вашим файлам. | • Оно не всегда доступно, потому что пользователь может монтировать external storage как USB MSD storage (Android работает как флешка, подключенная к компьютеру), и в некоторых случаях может просто извлечь external storage из устройства. • External storage доступно на чтение для всех, так что файлы, сохраненные туда, могут прочитаны без Вашего контроля над процессом. • Приложению требуется получить разрешение на доступ к external storage в файле манифеста. • Когда пользователь деинсталлирует Ваше приложение, система удалит файлы Вашего приложения из external storage только если Вы сохранили их в директории из getExternalFilesDir().  Вывод: external storage лучшее место для файлов, которые не требуют ограничений на доступ и для файлов, которые Вы хотите сделать общими с другими приложениями, или если Вы хотите, чтобы пользователь мог получить доступ к файлам с помощью компьютера. |   Совет: несмотря на то, что приложения по умолчанию устанавливаются в internal storage, Вы можете указать атрибутandroid:installLocation в файле манифеста, после чего Ваше приложение может быть установлено и на external storage. Пользователи ценят эту опцию, когда размер APK очень велик, и размер external storage space больше, чем internal storage. Дополнительную информацию см. в документации App Install Location [2].  [**Получение разрешения для приложения на доступ к External Storage**]  Чтобы иметь возможность записи в external storage, Вы должны запросить в файле манифеста разрешениеWRITE\_EXTERNAL\_STORAGE:  **< manifest** ... **>**  **< uses-permission** android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" **/>**  ...  **< /manifest >**  Внимание: в настоящее время приложения имеют возможность чтения external storage без специального на то разрешения. Однако это изменится в будущих релизах системы Android. Если Ваше приложение требует чтения external storage (но не записывает в него), то Вам нужно декларировать разрешение READ\_EXTERNAL\_STORAGE. Чтобы обеспечить будущую работу Вашего приложения так, как это ожидалось, Вы должны декларировать это разрешение уже сейчас, до того как изменения вступят в реальную силу.  **< manifest** ... **>**  **< uses-permission** android:name="android.permission.READ\_EXTERNAL\_STORAGE" **/>**  ...  **< /manifest >**  Но если Ваше приложение использует разрешение WRITE\_EXTERNAL\_STORAGE, то это неявно дает ему также разрешение использовать и чтение external storage.  Вам не нужно получать никаких разрешений на сохранение файлов в internal storage. Ваше приложение всегда имеет разрешение на чтение и запись файлов в свой внутренний каталог на internal storage.  [**Сохранение файла в Internal Storage**]  Когда сохраняется файл в internal storage, Вы можете запросить подходящую директорию для объекта файла File вызовом одного из двух методов:  getFilesDir() возвращает объект File, представляющий внутренний каталог Вашего приложения. getCacheDir() возвращает объект File, представляющий внутренний каталог временных файлов кэша Вашего приложения. Обязательно удаляйте оттуда каждый файл, когда он больше не нужен, и реализуйте разумный предел размера для объема памяти, который используете в любой момент времени, такой как предел в 1 мегабайт. Если система Android обнаружит, что на внутреннем хранилище недостаточно места, то она может удалить Ваши файлы кэша без предупреждения.  Чтобы создать новый файл в одной из этих директорий, Вы можете использовать конструктор File(), передав ему File, предоставленный одним из этих методов, которые укажут каталог на internal storage. Пример:  File file = **new** File(context.getFilesDir(), filename);  Альтернативно Вы можете вызвать openFileOutput(), чтобы получить FileOutputStream, который записывает файл в Вашей внутренней директории. Например, здесь показано, как записать некий текст в файл:  String filename = "myfile";  String string = "Hello world!";  FileOutputStream outputStream;  **try**  {  outputStream = openFileOutput(filename, Context.MODE\_PRIVATE);  outputStream.write(string.getBytes());  outputStream.close();  }  **catch** (Exception e)  {  e.printStackTrace();  }  Или, если Вам нужно кэшировать некоторые файлы, Вы должны вместо этого использовать createTempFile(). Например, следующий метод вытаскивает имя файла из [URL](http://developer.android.com/reference/java/net/URL.html) и создает файл с таким именем во внутренней директории для кэша Вашего приложения:  **public** File getTempFile(Context context, String url)  {  File file;  **try**  {  String fileName = Uri.parse(url).getLastPathSegment();  file = File.createTempFile(fileName, **null**, context.getCacheDir());  }  **catch** (IOException e)  {  *// Ошибка при создании файла*  ...  }  **return** file;  }  Примечание: каталог internal storage Вашего приложения указывается на основе имени пакета приложения в специальном месте файловой системы Android. Технически другое приложение может прочитать Ваши внутренние файлы, если Вы установите файловый режим с разрешенным чтением. Однако для этого другое приложение должно также знать имя пакета Вашего приложения и имена используемых Вашим приложением файлов. Другие приложения не могут просматривать Ваши внутренние директории, и не могут получить доступ на чтение или запись, за исключением случая, когда Вы явно установите файл как читаемый и/или записываемый. Таким образом, пока Вы используете MODE\_PRIVATEдля Ваших файлов на internal storage, то они никогда не будут доступны для других приложений.  [**Сохранение файла в External Storage**]  Поскольку external storage иногда может быть недоступно (когда пользователь смонтировал его как внешний USB-носитель на PC, или когда вытащил карту SD из телефона), то перед доступом к тому Вы должны всегда проверить, что он есть в наличии. Вы можете запросить состояние external storage вызовом getExternalStorageState(). Если возвращенное состояние External Storage равно MEDIA\_MOUNTED, то Вы можете читать и записывать на него свои файлы. Например, следующие методы полезны для определения доступности устройства хранения:  */\* Проверяет, доступно ли external storage для чтения и записи \*/*  **public** boolean isExternalStorageWritable()  {  String state = Environment.getExternalStorageState();  **if** (Environment.MEDIA\_MOUNTED.equals(state))  {  **return** **true**;  }  **return** **false**;  }    */\* Проверяет, доступно ли external storage как минимум для чтения \*/*  **public** boolean isExternalStorageReadable()  {  String state = Environment.getExternalStorageState();  **if** (Environment.MEDIA\_MOUNTED.equals(state) ||  Environment.MEDIA\_MOUNTED\_READ\_ONLY.equals(state))  {  **return** **true**;  }  **return** **false**;  }  Несмотря на то, что external storage может быть модифицировано пользователем и другими приложениями, есть две категории файлов, которые могут быть сохранены здесь:  Публичные файлы (Public files) - это файлы, которые должны быть свободно доступны для других приложений и пользователя. Когда пользователь деинсталлирует Ваше приложение, эти файлы должны остаться доступными для пользователя. Например, такими файлами могут быть фотографии, созданные другими приложениями или другие файлы, загруженные через сеть.  Частные файлы (Private files) - это файлы, полные права на которые принадлежат Вашему приложению, и которые должны быть удалены при деинсталляции Вашего приложения пользователем. Несмотря на то, что эти файлы технически доступны для пользователя и других приложений, поскольку они находятся на внешнем извлекаемом хранилище (external storage), эти файлы не имеют в реальности особого значения для пользователя вне Вашего приложения. Когда пользователь деинсталлирует Ваше приложение, система удалит все файлы в Вашем частном каталоге на внешнем хранилище. Примером таких файлов могут быть дополнительные ресурсы, загруженные Вашим приложением или временные медиафайлы.  Если Вы хотите сохранить public-файлы на external storage, используйте метод getExternalStoragePublicDirectory() для получения экземпляра File, предоставляющего подходящую директорию на external storage. Метод принимает аргумент, указывающий тип файла, который Вы хотите сохранить, так чтобы типы файла были логически организованы с другими public-файлами, такими как DIRECTORY\_MUSIC или DIRECTORY\_PICTURES. Пример:  **public** File getAlbumStorageDir(String albumName)  {  *// Получение каталога для публичного каталога картинок пользователя.*  File file = **new** File(Environment.getExternalStoragePublicDirectory(  Environment.DIRECTORY\_PICTURES), albumName);  **if** (!file.mkdirs())  {  Log.e(LOG\_TAG, "Директория не создана");  }  **return** file;  }  Если Вы хотите сохранить файлы, которые являются частными (private) для Вашего приложения, Вы можете получить подходящую директорию вызовом метода getExternalFilesDir() и передачей ему имени, указывающего тип директории, который Вам нужен. Каждая директория, созданная таким способом, будет добавлена к родительской директории, в которой инкапсулированы все файлы внешнего хранилища Вашего приложения, которые система удалит, когда пользователь деинсталлирует Ваше приложение. Например, вот метод, которым Вы можете создать директорию индивидуального фотоальбома:  **public** File getAlbumStorageDir(Context context, String albumName)  {  *// Получение каталога для приватного каталога картинок приложения.*  File file = **new** File(context.getExternalFilesDir(  Environment.DIRECTORY\_PICTURES), albumName);  **if** (!file.mkdirs())  {  Log.e(LOG\_TAG, "Директория не создана");  }  **return** file;  }  Если ни одно из предварительно определенных имен поддиректорий не подходит для Ваших файлов, то Вы можете вместо этого вызвать getExternalFilesDir() и передать null. Это возвратит корневую частную директорию для Вашего приложения на external storage.  Помните, что getExternalFilesDir() создает директорию внутри директории, которая будет удалена при деинсталляции Вашего приложения. Если файлы, которые Вы сохраняете, должны оставаться доступными после того, как пользователь деинсталлирует Ваше приложение - как например если Ваше приложение работает с фотокамерой, и пользователь хотел бы сохранить сделанные фотографии — Вы должны вместо этого использовать getExternalStoragePublicDirectory().  Независимо от того, используете ли Вы для публичных файлов getExternalStoragePublicDirectory() или getExternalFilesDir() для частных файлов приложения, важно иметь в виду, что Вы используете имена директорий, предоставленные константами API наподобие DIRECTORY\_PICTURES. Эти имена директорий гарантируют, что система будет правильно рассматривать эти файлы. Например файлы, сохраненные в DIRECTORY\_RINGTONES, будут рассортированы медиасканером системы как рингтоны вместо музыки.  [**Опрос количества свободного места**]  Если Вы знаете заранее, сколько файлов сохраняете, то можете без получения ошибок IOException узнать, сколько места осталось путем вызова getFreeSpace() или getTotalSpace(). Эти методы предоставляют соответственно текущее доступное пространство и общее пространство на томе хранения. Эта информация также полезна, чтобы избежать переполнения тома хранения свыше определенного порога.  Однако система не гарантирует, что Вы можете записать столько байт, сколько показывает вызов getFreeSpace(). Если возвращенное количество всего на несколько мегабайт больше, чем Вам нужно сохранить, или если файловая система уже заполнена меньше, чем на 90%, то вероятно сохранение будет безопасным. Иначе возможно, что записать данные в хранилище не получится.  Внимание: Вам не обязательно проверять количество свободного места перед сохранения файла. Вместо этого Вы можете попробовать записать файл сразу же, и затем перехватить исключение IOException, если оно произойдет. Вы возможно, должны так поступить, когда не знаете, сколько места Вам нужно. Например, если Вы меняете способ кодирования файла перед его сохранением, преобразовывая картинку PNG в JPEG, то Вы не будете знать размер файла заранее.  [**Удаление файла**]  Вы всегда должны удалять файлы, которые Вам больше не нужны. Самый прямой способ удаления файла состоит в том, чтобы иметь этот файл открытым и вызвать delete() для самого себя.  myFile.delete();  Если файл сохранен на internal storage, Вы можете также запросить Context, чтобы найти и удалить файл вызовом deleteFile():  myContext.deleteFile(fileName);  Внимание: когда пользователь деинсталлирует Ваше приложение, система Android удалит следующее:  • Все файлы, сохраненные Вашим приложением на internal storage. • Все файлы, сохраненные Вашим приложением с использованием getExternalFilesDir().  Однако Вы должны регулярно удалять все кэшируемые файлы, создаваемые с getCacheDir(), и также регулярно удалять файлы, которые Вам больше не нужны.  [**Пример записи файла на sdcard0**]  Предположим, что необходимо записать какой-нибудь тестовый файл (с именем myFile.txt) в папку myFolder на внешний носитель, который виден в системе Android как sdcard0. Т. е. полный путь должен выглядеть примерно так:  /basePath/myFolder/myFile.txt  **Базовый путь до External Storage**  Проблема тут состоит в том, чтобы узнать часть пути basePath, поскольку на разных системах Android этот путь будет разным, в зависимости от версии и внутреннего аппаратного устройства. В моем телефоне Samsung Galaxy Note этот basePath = /storage/sdcard0, но это еще не значит, что на Вашем телефона этот путь будет именно таким. Чтобы получить basePath, используйте вызов функции getExternalStorageDirectory:  *//Вызов getExternalStorageDirectory вернет путь до "внешнего" (External)*  *// носителя, например /storage/sdcard0*  String basePath = Environment.getExternalStorageDirectory().getAbsolutePath();  **Проверка доступности носителя данных в External Storage**  Вторая проблема состоит в доступности на запись носителя данных. Дело в том, что записать на носитель можно не всегда, например если он смонтирован как флешка USB (когда Ваш телефон подключен к компьютеру в режиме Mass Storage Device, USB MSD). Проверить доступность носителя можно следующей функцией:  */\* Проверяет, доступно ли external storage для чтения и записи \*/*  **public** boolean isExternalStorageWritable()  {  String state = Environment.getExternalStorageState();  **if** (Environment.MEDIA\_MOUNTED.equals(state))  {  **return** **true**;  }  **return** **false**;  }  **Разрешение доступа к носителю данных в файле манифеста**  Как уже упоминалось, необходимо в файле манифеста запросить разрешение WRITE\_EXTERNAL\_STORAGE. Вот пример такого файла манифеста:  < ?xml version="1.0" encoding="utf-8"? >  **< manifest** xmlns:android="http://schemas.android.com/apk/res/android"  package="com.findusbdevbr"  android:versionCode="1"  android:versionName="1.0" **>**  **< uses-feature** android:name="android.hardware.usb.host" **/>**  **< uses-permission** android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" **/>**  **< uses-sdk**  android:minSdkVersion="13"  android:targetSdkVersion="19" **/>**  **< application**  android:allowBackup="true"  android:icon="@drawable/ic\_launcher"  android:label="@string/app\_name"  android:theme="@style/AppTheme" **>**  **< activity**  android:name="com.findusbdevbr.MainActivity"  android:label="@string/app\_name" **>**  **< intent-filter>**  **< action** android:name="android.intent.action.MAIN" **/>**  **< category** android:name="android.intent.category.LAUNCHER" **/>**  **< /intent-filter >**  **< /activity >**  **< /application >**  **< /manifest >**  Функция, которая сохраняет файл, принимая полный путь до файла filePath и сохраняемый текст FileContent:  **public** void SaveFile (String filePath, String FileContent)  {  *//Создание объекта файла.*  File fhandle = **new** File(filePath);  **try**  {  *//Если нет директорий в пути, то они будут созданы:*  **if** (!fhandle.getParentFile().exists())  fhandle.getParentFile().mkdirs();  *//Если файл существует, то он будет перезаписан:*  fhandle.createNewFile();  FileOutputStream fOut = **new** FileOutputStream(fhandle);  OutputStreamWriter myOutWriter = **new** OutputStreamWriter(fOut);  myOutWriter.write(FileContent);  myOutWriter.close();  fOut.close();  }  **catch** (IOException e)  {  *//e.printStackTrace();*  textInfo.setText("Path " + filePath + ", " + e.toString());  }  }  Вызов функции SaveFile, который выполняет задачу сохранения файла в External-носителе:  **public** void btnClick(View v)  {  String fullpath, foldername, filename;  foldername = "temp/myFolder";  filename = "myFile.txt";    *//Сохранение файла на External Storage:*  fullpath = Environment.getExternalStorageDirectory().getAbsolutePath()  + "/" + foldername  + "/" + filename;  **if** (isExternalStorageWritable())  {  SaveFile(fullpath, "Этот текст сохранен на External Storage");  }  }  [**Пример записи файла на extSdCard**]  Получение полного корневого пути до извлекаемой карты SD не так прост, как до External Storage, поскольку в API Android для этого почему-то не предусмотрены специальные простые функции. Приходится получать путь окольными путями, через имена системных папок. Вот код функции, которая получает путь до извлекаемой карты SD:  *//Функция определяет путь до внешней извлекаемой карты*  *// наподобие /storage/extSdCard*  **private** String getSDcardPath()  {  String exts = Environment.getExternalStorageDirectory().getPath();  String sdCardPath = **null**;  **try**  {  FileReader fr = **new** FileReader(**new** File("/proc/mounts"));  BufferedReader br = **new** BufferedReader(fr);  String line;  **while**((line = br.readLine())!=**null**)  {  **if**(line.contains("secure") || line.contains("asec"))  **continue**;  **if**(line.contains("fat"))  {  String[] pars = line.split("\\s");  **if**(pars.length<2)  **continue**;  **if**(pars[1].equals(exts))  **continue**;  sdCardPath =pars[1];  **break**;  }  }  fr.close();  br.close();  **return** sdCardPath;  }  **catch** (Exception e)  {  *//e.printStackTrace();*  textInfo.setText(e.toString());  }  **return** sdCardPath;  }  Вызов функции SaveFile, который выполняет задачу сохранения файла на извлекаемой карте SD:  **public** void btnCheckUsbDevClick(View v)  {  String fullpath, foldername, filename;  foldername = "temp/myFolder";  filename = "myFile.txt";    *//Сохранение файла на карту SD:*  fullpath = getSDcardPath()  + "/" + foldername  + "/" + filename;  SaveFile(fullpath, "Этот текст сохранен на карту SD");  }  [**Сохранение бинарного файла (массива byte[])**]  В предыдущих примерах мы рассматривали класс OutputStreamWriter, который позволяет записать строку String или массив символов char[]. Но как быть, если нужно записать массив байт byte[]? Для этого подойдет классDataOutputStream. Пример:  **public** **static** void SaveBin (byte[] buf, int count)  {  String foldername = "temp/myfolder";  String filename = "myfile.bin";  String fullpath = fileutil.getSDcardPath() + "/" + foldername + "/" + filename;  *//Создание объекта файла.*  File fhandle = **new** File(fullpath);  **try**  {  *//Если нет директорий в пути, то они будут созданы:*  **if** (!fhandle.getParentFile().exists())  fhandle.getParentFile().mkdirs();  *//Если файл существует, то он будет перезаписан:*  fhandle.createNewFile();  FileOutputStream fOut = **new** FileOutputStream(fhandle);  DataOutputStream myOutWriter = **new** DataOutputStream(fOut);  myOutWriter.write(buf, 0, count);  myOutWriter.flush();  myOutWriter.close();  fOut.close();  }  **catch** (IOException e)  {  Toast.makeText(context,  "SaveFile: path " + fullpath + ", " + e.toString(),  Toast.LENGTH\_LONG).show();  }  }  Примеры вызовов getAbsolutePath:   |  |  | | --- | --- | | **Вызов** | **Результат вызова** | | Environment.getRootDirectory.getAbsolutePath() | /system | | Environment.getExternalStorageDirectory().getAbsolutePath() | /storage/sdcard0 | | Environment.getExternalStoragePublicDirectory(null).getAbsolutePath() | завершится с ошибкой | | Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY\_ALARMS).getAbsolutePath() | /storage/sdcard0/Alarms |   [**Ссылки**]  **1**. Saving Files site:developer.android.com. **2**. App Install Location site:developer.android.com. |