**Лекция 9. Поставщики и загрузчики**

В этой лекции обсуждается, как получить доступ к данным из **Content Provider** с помощью [**Loader**](https://developer.android.com/guide/components/loaders.html) . *Content Provider* — это абстракция базы данных или другого хранилища данных, позволяющая нам легко и систематически работать с этими данными в Java (а не на отдельном языке манипулирования данными, таком как SQL). Затем *Loader* используется для эффективного выполнения этого доступа к данным в фоновом режиме (вне потока пользовательского интерфейса), а также легкого подключения этих данных к представлениям. В этой лекции обсуждается, как использовать Loader для доступа к данным в существующем Content Provider; в [более поздней лекции](https://info448.github.io/databases.html#databases) подробно описывается, как создавать Content Providers с нуля.

В этой лекции используется код, найденный по адресу <https://github.com/info448/lecture09-loaders> . Обратите внимание, что эта демонстрация обращается к пользовательскому словарю устройства, который доступен только для общих приложений на API 22 (Lollipop) **или более ранних версиях** . Таким образом, это руководство не поддерживает все версии Android. Вместо этого вы можете использовать эмулятор, работающий на API 22 или более ранних версиях.

**9.1 Поставщики контента**

В примере начального кода используется , ListViewкоторый показывает список слов. Вспомните, что , ListViewиспользует архитектуру **модель-представление-контроллер** … и в этом случае «модель» (данные) — это жестко закодированный список массива слов. Но есть и другие списки слов! Целые базы данных слов! В предыдущих лекциях обсуждалось, как использовать *сетевые запросы* для доступа к API онлайн-данных, но есть также базы данных (слов, не меньше), встроенные в ваш телефон Android.

Например, Android отслеживает список написаний «нестандартных» слов в так называемом **Пользовательском словаре** . Вы можете просмотреть этот список на устройстве по адресу Settings > Language & Input > Personal Dictionary. Вы даже можете использовать этот интерфейс настроек для добавления новых слов в словарь (например, «embiggen», «cromulent», «covfefe»).

Из соображений безопасности пользовательский словарь доступен только на устройствах с **API 22 (Lollipop)** или более ранней версии\*\*. Этот конкретный пример будет работать только на более старых устройствах, но концепции и код для доступа к базам данных в целом будут применяться ко всем версиям. Вы можете протестировать этот конкретный пример с помощью эмулятора с API 22 или более ранней версии.

Обратите внимание, что пользовательский словарь отслеживает [**базу**](https://developer.android.com/guide/topics/providers/content-provider-basics.html#ClientProvider) данных слов. Вы можете представить эту базу данных как единую таблицу SQL (или электронную таблицу CSV, или фрейм данных R): это набор *записей* (строк), каждая из которых имеет некоторые *значения* (столбцы). Первичный ключ таблицы называется (по соглашению) **ID**.

Поскольку эти данные хранятся в (по сути) простой таблице SQL, мы можем получить к ним доступ и изменить их программно&mash;более того, фреймворк Android позволяет нам делать это без необходимости знать или писать SQL! Например, мы можем получить доступ к этому списку слов, чтобы отобразить их в ListView.

Хотя вам не обязательно знать SQL для использования встроенной базы данных, такой как Пользовательский словарь, будет полезно иметь поверхностное знакомство с реляционными базами данных и их терминологией, чтобы иметь интуитивное представление об организации данных.

Для этого нам нужно запросить разрешение на доступ к базе данных, так же, как мы запросили разрешение на доступ к Интернету. Включите в *Manifest* следующее :

**<uses-permission** android:name="android.permission.READ\_USER\_DICTIONARY"**/>**

Хотя слова хранятся в базе данных, мы не знаем *точного* формата этой базы данных (например, точные имена таблиц или столбцов, или даже является ли это базой данных SQL или просто файлом .csv!). Мы хотим избежать необходимости писать код, который работает только с определенным форматом, особенно потому, что слова могут храниться в различных типах баз данных на разных устройствах или в разных версиях Android. (Фреймворк Android включает поддержку работы напрямую с локальной SQLiteбазой данных, как обсуждается в следующей главе).

Чтобы не полагаться на конкретный формат хранения некоторых данных, Android предлагает ***абстракцию*** в виде [**Content Provider**](https://developer.android.com/guide/topics/providers/content-providers.html)[33](https://info448.github.io/loaders.html#fn33) . Content Provider предлагает интерфейс для взаимодействия со структурированными данными, независимо от того, хранятся ли эти данные в базе данных, в файле, в *нескольких* файлах, в сети или где-то еще. Таким образом, вы можете думать о «Content Provider» как о «источнике данных» (например, источник/поставщик контента)!

* Хорошей метафорой будет представление о поставщике контента как о «сервере», к которому вы получаете доступ.
* Можно создавать собственных поставщиков контента (описано в следующей главе), но данная лекция посвящена исключительно *использованию* существующих поставщиков.

Все поставщики контента (источники данных) имеют **URI** ( [универсальный идентификатор ресурса](https://en.wikipedia.org/wiki/Uniform_Resource_Identifier) , обобщение URL, используемого для ресурсов, не обязательно в Интернете). Можно *запросить* этот URI, аналогично тому, как веб-API получают доступ через запросы к своим конечным точкам URI. В частности, URI поставщиков контента используют протокол **content://**(вместо https://), поскольку доступ к их данным осуществляется через «запросы контента», а не «запросы HTTP».

Для содержимого URIсловаря определяется константой [UserDictionary.Words.CONTENT\_URI](https://developer.android.com/reference/android/provider/UserDictionary.Words.html). Мы используем константы для ссылки на URI и пути, чтобы упростить ссылку на них и обобщение для устройств, которые могут иметь разные структуры каталогов.

Мы можем получить доступ к этому Content Provider через [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html). Этот класс предоставляет методы для доступа к данным в поставщике (представленном как ContentProviderобъект). Каждый Context имеет синглтон ContentResolver, доступ к которому осуществляется через getContentResolver()метод (обратите внимание, что для Fragment Context является содержащей Activity). Методы ContentResolver поддерживают основные операции [CRUD](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete)insert() : (создание), query()(чтение), update()и delete().

*//java*

ContentResolver resolver = getContentResolver();

Методы ContentResolver принимают несколько параметров, поддерживая различные опции, доступные в общем запросе SQL. Например, рассмотрим метод [query()](https://developer.android.com/reference/android/content/ContentResolver.html#query(android.net.Uri,%20java.lang.String[],%20java.lang.String,%20java.lang.String[],%20java.lang.String)):

getContentResolver().query(

uri, *// The content URI*

projection, *// The an array of columns to return for each row*

selectionClause *// Selection criteria (as an SQL WHERE clause)*

selectionArgs, *// An array of values that can be injected into the selection clause*

sortOrder); *// The sort order for the returned rows (as an SQL ORDER BY clause)*

* По сути, это оболочка вокруг SELECTоператора SQL! Но каждая «часть» этого оператора указывается как параметр этого метода.

Проекция **—** это String[]все «столбцы» (атрибуты), которые мы хотим получить из источника данных. Это то, что вы бы поставили после SELECTв SQL. (Обратите внимание, что мы можем передать nullдля представления SELECT \*, но это неэффективно — лучше дать список всего).

* Мы можем увидеть, какие имена столбцов доступны для пользовательского словаря в [UserDictionary.Words](http://developer.android.com/reference/android/provider/UserDictionary.Words.html). Опять же, они определены как константы!
* Обязательно всегда выбирайте \_IDпервичный ключ: он понадобится позже!

Другие параметры могут использоваться для настройки SELECTоператора. Предложение «выбора» ( WHERE) требует параметров: вторые — это значения, которые будут экранированы от [атак SQL-инъекций](https://en.wikipedia.org/wiki/SQL_injection) . Передача nullлюбого из этих параметров приведет к игнорированию предложения:

*//java*

String[] projection = **new** String[] {

UserDictionary.Words.WORD,

UserDictionary.Words.FREQUENCY,

UserDictionary.Words.\_ID

};

resolver.query(UserDictionary.Words.CONTENT\_URI, projection, **null**, **null**, **null**);

*//kotlin*

**val** projection = arrayOf(

UserDictionary.Words.WORD,

UserDictionary.Words.FREQUENCY,

UserDictionary.Words.\_ID

)

contentResolver.query(UserDictionary.Words.CONTENT\_URI, projection, **null**, **null**, **null**)

Итак, в целом, запрос разбивает SQL- SELECTвыражение компонентов на разные части в качестве параметров метода, так что вам не *обязательно* писать выборку самостоятельно. Более того, этот метод *абстрагирует* конкретный язык запросов, позволяя использовать одни и те же запросы в разных форматах баз данных ( SQLite, PostgreSQL, файлы и т. д.).

**9.2 Курсоры**

Метод ContentResolver#query()возвращает [**Cursor**](http://developer.android.com/reference/android/database/Cursor.html). A Cursorпредоставляет интерфейс для списка записей в базе данных (например, тех, которые возвращаются запросом). A Cursorтакже ведет себя как Iteratorв Java: он отслеживает, к какой записи в данный момент осуществляется доступ (например, что iбудет в цикле for). Вы можете думать о нем как об «указателе» на определенную запись, как курсор на экране.

Мы вызываем методы для Cursorуказания, на какую запись мы хотим, чтобы он «указывал», а также для извлечения значений из объекта записи в этом месте списка. Например:

*//java*

cursor.moveToFirst(); *//move to the first item*

String field0 = cursor.getString(0); *//get the first field (column you specified) as a String*

String word = cursor.getString(cursor.getColumnIndexOrThrow("word")); *//get the "word" field as a String*

cursor.moveToNext(); *//go to the next item*

*//kotlin*

cursor.moveToFirst() *//move to the first item*

**val** field0 = cursor.getString(0) *//get the first field (column you specified) as a String*

**val** word = cursor.getString(cursor.getColumnIndexOrThrow("word")) *//get the "word" field as a String*

cursor.moveToNext() *//go to the next item*

Однако приятно Cursorsто, что их можно легко вводить AdapterViewsс помощью [CursorAdapter](http://developer.android.com/reference/android/widget/CursorAdapter.html)(в отличие от , который ArrayAdapterмы использовали ранее). Это [**SimpleCursorAdapter**](http://developer.android.com/reference/android/widget/SimpleCursorAdapter.html)конкретная реализация, которая почти так же проста в использовании, как и ArrayAdapter.

Вы создаете новый экземпляр SimpleCursorAdapter, передавая ему:

1. A Contextдля загрузки ресурсов
2. Ресурс макета для наполнения каждой записи
3. А Cursor(что может быть null)
4. Массив имен столбцов для извлечения из каждой записи в курсоре ( **проекция** , аналогичная предыдущей)
5. Соответствующий список ресурсов View ids(которые все должны быть TextViews) для назначения значения каждого столбца. Это «отображение», которое будет выполнять адаптер (из столбцов проекции в содержимое TextView).
6. Любые дополнительные опции флаги ( 0означает отсутствие флагов и является правильным вариантом для нас).

*//java*

adapter = **new** SimpleCursorAdapter(

**this**,

R.layout.list\_item\_layout, *//item to inflate*

cursor, *//cursor to show*

**new** String[] {UserDictionary.Words.WORD, UserDictionary.Words.FREQUENCY}, *//fields to display*

**new** int[] {R.id.txt\_list\_item, R.id.txt\_item\_freq}, *//where to display them*

0); *//flags*

*//kotlin*

adapter = SimpleCursorAdapter(

**this**,

R.layout.list\_item\_layout, *//item to inflate*

cursor, *//cursor to show*

arrayOf(UserDictionary.Words.WORD, UserDictionary.Words.FREQUENCY), *//fields to display*

intArrayOf(R.id.txt\_item\_word, R.id.txt\_item\_freq), *//where to display them*

0 *//flags*

)

Затем мы можем использовать этот адаптер для ListView вместо ArrayAdapter!

**9.3 Грузчики**

Чтобы передать Cursorв адаптер, нам нужна .query()база данных. Но мы хотим делать это очень часто, каждый раз, когда база данных обновляется, мы хотели бы иметь возможность снова запрашивать ее, чтобы обновить адаптер и увидеть изменения! Кроме того, доступ к базе данных может быть *медленным* (требуется доступ к диску, структурирование и отправка вызовов SQL, и в зависимости от сложности базы данных эти запросы могут занять время). Таким образом, как и в случае с сетевыми запросами, мы хотели бы выполнить этот запрос в *фоновом потоке* , чтобы он не блокировал наше приложение и не приводил к его остановке.

Чтобы автоматически обновлять ваш список новыми данными, загруженными в фоновый поток, мы будем использовать класс с именем [Loader](https://developer.android.com/guide/components/loaders.html). По сути, это оболочка вокруг [ASyncTask](https://developer.android.com/reference/android/os/AsyncTask.html)(описанная в следующей главе), но она позволяет вам многократно выполнять фоновую задачу *при изменении источника данных* . В частности, Android предоставляет [CursorLoader](http://developer.android.com/reference/android/content/CursorLoader.html)специально используемый для загрузки данных из ContentProviders через Cursors — всякий раз, когда изменяется содержимое, создается новый Cursor, который можно «подставить» в адаптер.

Загрузчики устарели в Android P (API 28) в пользу классов пакета «Architecture», таких как ViewModelи LiveData. Но многие из тех же идей будут применяться.

Чтобы использовать CursorLoader, нам нужно указать, что наша Activity реализует [LoaderManager.LoaderCallbacks<Cursor>](https://developer.android.com/reference/android/support/v4/app/LoaderManager.LoaderCallbacks.html)интерфейс — по сути, это означает, что этот Fragment может реагировать на события Loader.

Для использования нам потребуется заполнить функции обратного вызова интерфейсов CursorLoader:

* В мы указываем, что должен *делать*onCreateLoader() Loader . Здесь мы создадим экземпляр и вернем , который запрашивает . Это очень похоже на метод, который мы написали ранее, но будет работать в фоновом потоке![new CursorLoader(...)](https://developer.android.com/reference/android/content/CursorLoader.html#CursorLoader(android.content.Context,%20android.net.Uri,%20java.lang.String[],%20java.lang.String,%20java.lang.String[],%20java.lang.String))ContentProvider.query()
* В onLoadFinished()обратном вызове мы можем использовать swapCursor()для обмена переданными в Cursorнаш SimpleCursorAdapter, чтобы передать эти данные модели в наш контроллер (для отображения в представлении). Фреймворк обрабатывает любую очистку вокруг старого Курсора.
* В onLoaderReset()обратном вызове мы можем просто заменить nullнаш курсор, поскольку теперь нет контента для отображения (загруженные данные были «сброшены»).

Наконец, чтобы фактически *начать* нашу фоновую загрузку, мы используем getLoaderManager().initLoader(...)метод. Это заставит фреймворк Android запросить создание нового Loader (нашим onCreateLoader()методом), а также начать загрузку этого Loader! (Это использует менеджер, похожий на FragmentManager, и по вкусу похож на AsyncTask.execute()).

getSupportLoaderManager().initLoader(0, **null**, **this**);

* Используйте , getSupportLoaderManager()если вы используете библиотеку поддержки (и вызываете из Activity; фрагмент библиотеки поддержки, который мы использовали, имеет только одного менеджера, поэтому вы можете просто использовать getLoaderManager()).

Первый параметр метода initLoader()— это номер идентификатора, для *которого вы хотите загрузить курсор* — то, что передается в качестве первого параметра onCreateLoader()(или доступно через Loader#getId()). Это позволяет вам иметь несколько загрузчиков, использующих одну и ту же функцию обратного вызова (например, для обработки нескольких загрузчиков для нескольких источников данных). Второй параметр — это Bundleargs, а третий — LoaderCallbacks(например, кто обрабатывает результаты)!

* Обратите внимание, что вы можете использовать этот .restartLoader()метод для «воссоздания» CursorLoader(без потери других ссылок), например, если вы хотите изменить переданные ему аргументы.

И с помощью этого мы можем извлекать слова из нашей базы данных в фоновом режиме, и если мы обновим слова (например, через языковые настройки), они автоматически обновятся!

**9.4 Другие действия поставщика**

Конечно, Content Resolver позволяет нам делать больше, чем просто запрашивать и загружать данные: мы также можем добавлять, обновлять или удалять записи из базы данных.

* Если мы хотим *изменить* содержимое пользовательского словаря, нам понадобится разрешение:

**<uses-permission** android:name="android.permission.WRITE\_USER\_DICTIONARY"**/>**

Чтобы *вставить* (создать) новое слово в ContentProvider, мы вызываем .insert()метод в ContentResolver. Ему передается [ContentValues](http://developer.android.com/reference/android/content/ContentValues.html)объект, который является HashMap, почти таким же, как Bundle(но он поддерживает только значения, которые могут быть введены в ContentProvider, например, Parcelables не поддерживаются).

*//java*

ContentValues newValues = **new** ContentValues();

newValues.put(UserDictionary.Words.WORD, inputText.getText().toString());

newValues.put(UserDictionary.Words.FREQUENCY, 100);

newValues.put(UserDictionary.Words.APP\_ID, "edu.uw.loaderdemo");

newValues.put(UserDictionary.Words.LOCALE, "en\_US");

Uri newUri = getContentResolver().insert(

UserDictionary.Words.CONTENT\_URI, *// the user dictionary content URI!*

newValues *// the values to insert*

);

*//kotlin*

**val** newValues = ContentValues()

newValues.put(UserDictionary.Words.WORD, inputText.text.toString())

newValues.put(UserDictionary.Words.FREQUENCY, 100)

newValues.put(UserDictionary.Words.APP\_ID, "edu.uw.loaderdemo")

newValues.put(UserDictionary.Words.LOCALE, "en\_US")

**val** newUri = contentResolver.insert(

UserDictionary.Words.CONTENT\_URI, *// the user dictionary content URI!*

newValues *// the values to insert*

)

* Функция insert()возвращает URI для *вновь вставленной строки* , например, если вы хотите иметь возможность запросить и отобразить это содержимое позже.

Аналогичный подход используется для *обновления* и изменения записи в Content Provider: вызовите .update()метод и передайте ContentValuesнабор значений для изменения:

*//java*

ContentValues newValues = **new** ContentValues();

newValues.put(UserDictionary.Words.FREQUENCY, newFrequency);

getContentResolver().update(

ContentUris.withAppendedId(UserDictionary.Words.CONTENT\_URI, id),

newValues,

**null**, **null**); *//no selection*

*//kotlin*

**val** newValues = ContentValues()

newValues.put(UserDictionary.Words.FREQUENCY, newFrequency)

contentResolver.update(

ContentUris.withAppendedId(UserDictionary.Words.CONTENT\_URI, id),

newValues, **null**, **null** *//no selection*

)

* Обратите внимание, что мы гарантируем обновление только этого конкретного элемента, указав *URI этого элемента* . Мы делаем это, создавая новый URI, представляющий/идентифицирующий этот элемент, фактически добавляя "/:id"параметр пути к URI. Это означает, что нам не нужно использовать критерии выбора (хотя мы могли бы сделать и это).

Это охватывает то, как использовать и взаимодействовать с поставщиком контента (как *клиент* этого поставщика). [Следующая лекция](https://info448.github.io/databases.html#databases) будет посвящена тому, как реализовать поставщик для пользовательской базы данных.

1. <https://developer.android.com/guide/topics/providers/content-providers.html>[↩](https://info448.github.io/loaders.html#fnref33)