

# **ContentProvider** **(Поставщик контента)**

Поставщики контента управляют доступом к структурированному набору данных. Они инкапсулируют данные и предоставляют механизмы обеспечения их безопасности. Поставщики контента представляют собой стандартный интерфейс для объединения данных в одном процессе с кодом, который выполняется в другом процессе.

# ContentProvider

public abstract class ContentProvider  
extends [Object](#)

implements [ComponentCallbacks2](#)

Известные прямые подклассы:

[DocumentsProvider](#),

[MockContentProvider](#),

[SearchRecentSuggestionsProvider](#),

[SliceProvider](#)

# Что это и для чего?

Контент-провайдеры являются одним из основных строительными блоками приложений для Android, обеспечивая контент для приложений.

Они инкапсулируют данные и предоставляют их приложениям через интерфейс [ContentResolver](#).

Контент-провайдер требуется только в том случае, если вам нужно поделиться данные между несколькими приложениями.

Например, данные контактов используются несколькими пользователями приложения и должны храниться в контент-провайдере.

Если вам не нужно делиться данными между собой несколько приложений вы можете использовать базу данных непосредственно через [SQLiteDatabase](#).

# Как это работает?

Когда запрос делается через [ContentResolver](#) система проверяет полномочия данного URI и передает запрос к зарегистрированному поставщику контента.

Контент-провайдер может интерпретировать остальную часть URI, как он хочет.

[UriMatcher](#) класс полезен для синтаксического анализа идентификатора URI.

Основными методами, которые необходимо реализовать, являются:

[onCreate\(\)](#) который вызывается для инициализации поставщика

[query\(Uri, String\[\], Bundle, CancellationSignal\)](#) который возвращает данные вызывающему абоненту

[insert\(Uri, ContentValues\)](#) который вставляет новые данные в контент провайдера

[update\(Uri, ContentValues, Bundle\)](#) который обновляет существующие данные в поставщике контента

[delete\(Uri, Bundle\)](#) который удаляет данные из поставщика контента

[getType\(Uri\)](#) который возвращает тип MIME данных в поставщике контента.

# Потокобезопасность методов

Методы доступа к данным (такие как [insert\(Uri, ContentValues\)](#) и [update\(Uri, ContentValues, Bundle\)](#)) могут вызываться сразу из нескольких потоков и должны быть потокобезопасным.

Другие методы (например [onCreate\(\)](#)) вызывает только из приложения в основном потоке, и должны избегать выполнения длительных операций.

Запросы на [ContentResolver](#) -автоматически пересылаются в соответствующие Экземпляры **ContentProvider**, так что подклассам не нужно беспокоиться о деталях межпроцессных ВЫЗОВОВ.

# Доступ к данным в поставщике контента

Когда вам требуется доступ к данным в поставщике контента, используйте объект [ContentResolver](#) в интерфейсе [Context](#) вашего приложения, чтобы подключиться к поставщику как клиент.

Объект [ContentResolver](#) взаимодействует с объектом поставщика, который представляет собой экземпляр класса, реализующий объект [ContentProvider](#). Объект поставщика получает от клиентов запросы данных, выполняет запрашиваемые действия и возвращает результаты.

Вам не нужно разрабатывать собственный поставщик, если вы не планируете предоставлять доступ к своим данным другим приложениям.

- **Однако вам потребуется собственный поставщик для предоставления настраиваемых поисковых подсказок в вашем собственном приложении.**
- **Вам также потребуется собственный поставщик, если вы хотите копировать и вставлять сложные данные или файлы из своего приложения в другие приложения.**

# Поставщики контента в Андроид

В состав системы Android входят поставщики контента, которые управляют такими данными, как аудио, видео, изображения и личная контактная информация.

Некоторые из поставщиков указаны в справочной документации для пакета [android.provider](#) .

Работать с этими поставщиками может любое приложение Android (однако с некоторыми ограничениями).

# Основные сведения о поставщике контента

Поставщик контента управляет доступом к центральному репозиторию данных.

Поставщик является компонентом приложения Android, который зачастую имеет собственный пользовательский интерфейс для работы с данными.

Однако поставщики контента предназначены в первую очередь для использования другими приложениями, которые получают доступ к поставщику посредством клиентского объекта поставщика.

Вместе поставщики и клиенты поставщиков обеспечивают согласованный, стандартный интерфейс к данным, который также обрабатывает взаимодействие между процессами и обеспечивает защищенный доступ к данным.



# Рассматриваются основные сведения, касающиеся следующих тем:

- принцип работы поставщика контента;
- API, используемый для получения данных от поставщика контента;
- API, используемый для вставки данных в поставщик контента и их обновления или удаления в нем;
- другие функции API, которые упрощают работу с поставщиками.

# Обзор на примере

Поставщик контента предоставляет данные внешним приложениям в виде одной или нескольких таблиц, аналогичных таблицам в реляционной базе данных.

Строка представляет собой экземпляр некоторого типа собираемых поставщиком данных, а каждый столбец в этой строке — это отдельный элемент данных, собранных для экземпляра.

**Примером встроенного поставщика в платформе Android** может служить **пользовательский словарь**, в котором хранятся данные о написании нестандартных слов, добавленных пользователем.

В каждой строке таблицы представлен экземпляр слова, которое отсутствует в стандартном словаре.

В каждом ее столбце содержатся некоторые данные для слова, например, данные о языке, на котором это слово было впервые использовано.

Заголовки столбцов представляют собой имена столбцов, которые хранятся в поставщике.

Чтобы узнать язык строки, необходимо обратиться к столбцу `locale` (место действия).

В этом поставщике столбец `_ID` выступает в роли «основного ключа», который поставщик автоматически сохраняет.

В таблице показано, как данные могут выглядеть в этой таблице поставщика

слово	идентификатор приложения	частота	место действия	_ИДЕНТИФИКАТОР
mapreduce	пользователь1	100	en_US	1
прекомпилятор	пользователь14	200	fr_FR	2
апплет	пользователь2	225	fr_CA	3
константа	пользователь1	255	pt_BR	4
int	user5	100	en_UK	5

# Доступ к поставщику

Для доступа приложения к данным из поставщика контента используется клиентский объект [ContentResolver](#).

В этом объекте имеются методы, которые вызывают идентичные методы в объекте поставщика, который представляет собой экземпляр одного из конкретных подклассов класса [ContentProvider](#).

В этих методах [ContentResolver](#) представлены основные функции CRUD (аббревиатура create, retrieve, update, delete [создание, получение, обновление и удаление]) постоянного хранилища.

Объект [ContentResolver](#) в процессе клиентского приложения и объект [ContentProvider](#) в приложении, которое владеет поставщиком, автоматически обрабатывают взаимодействие между процессами.

Объект [ContentProvider](#) также выступает в роли уровня абстракции между репозиторием данных и внешним представлением данных в виде таблиц.

**Примечание.** Для доступа к поставщику ваше приложение обычно должно запросить определенные разрешения в своем файле манифеста. Дополнительные сведения об этом представлены в разделе [Разрешения поставщика контента](#).

# Пример

Например, чтобы получить из поставщика пользовательского словаря список слов и языков, на которых они представлены, вызовите метод [ContentResolver.query\(\)](#).

В свою очередь, метод [query\(\)](#) вызывает метод [ContentProvider.query\(\)](#), определенный поставщиком пользовательского словаря.

В примере кода ниже показан вызов метода [ContentResolver.query\(\)](#).

```
// Запрашивает пользовательский словарь и возвращает  
результаты
```

```
mCursor = getContentResolver().query(  
    UserDictionary.words.CONTENT_URI, // URI содержимого таблицы  
    СЛОВ  
    mProjection, // столбцы, возвращаемые для каждой  
    строки  
    mSelectionClause // критерии отбора  
    mSelectionArgs, // критерии отбора  
    mSortOrder); // порядок сортировки возвращаемых строк
```

В таблице указано соответствие аргументов для метода

[query\(Uri,projection,selection,selectionArgs,sortOrder\)](#)

SQL-инструкции  
SELECT.

Аргумент метода <code>query()</code>	Параметр/ключевое слово SELECT	Примечания
<code>Uri</code>	<code>FROM table_name</code>	<code>Uri</code> соответствует таблице <code>table_name</code> в поставщике.
<code>projection</code>	<code>col, col, col, ..</code>	<code>projection</code> представляет собой массив столбцов, которые следует включить в каждую полученную строку.
<code>selection</code>	<code>WHERE col = value</code>	<code>selection</code> задает критерии для выбора строк.
<code>selectionArgs</code>	(Точный эквивалент отсутствует. В предложении выбора заполнители ? заменяются аргументами выбора).	
<code>sortOrder</code>	<code>ORDER BY col, col, ...</code>	<code>sortOrder</code> задает порядок отображения строк в возвращаемом объекте <code>Cursor</code> .

# URI контента

**URI контента** представляет собой URI, который определяет данные в поставщике.

URI контента могут включать символическое имя всего поставщика (его **центр**) и имя, которое указывает на таблицу (**путь**).

При вызове клиентского метода для доступа к таблице в поставщике **URI** контента этой таблицы выступает в роли одного из аргументов этого метода.

Константа [CONTENT\\_URI](#) в предыдущих строках кода содержит URI контента таблицы words в пользовательском словаре.

Объект [ContentResolver](#) анализирует центр URI и использует его для «разрешения» поставщика путем сравнения центра с системной таблицей известных поставщиков.

[ContentResolver](#) может отправить аргументы запроса в соответствующий поставщик.

[ContentProvider](#) использует часть URI контента, в которой указан путь, для выбора таблицы для доступа.

В поставщике обычно имеется **путь** для каждой предоставляемой им таблицы.

В предыдущих строках кода полный URI для таблицы words выглядит следующим образом:

```
content:// user_dictionary / words
```

Строка user\_dictionary – это центр поставщика, а строка words — это путь к таблице.

Строка content:// (**схема**) присутствует всегда; она определяет, что это URI контента.

# URI контента

Многие поставщики предоставляют доступ к одной строке в таблице путем добавления идентификатора в конец URI.

Например, чтобы извлечь из пользовательского словаря строку, в столбце `_ID` которой указано 4, можно воспользоваться следующим URI контента:

```
Uri singleUri =  
    ContentUris.withAppendedId(UserDictionary.words.CONTENT_URI,4);
```

Идентификаторы часто используются в случае, когда вы извлекли набор строк и хотите обновить или удалить одну из них.

**Примечание.** В классах [Uri](#) и [Uri.Builder](#) имеются методы для удобного создания правильно оформленных объектов URI из строк.

[ContentUris](#) содержит методы для удобного добавления идентификаторов к URI.

В примере кода выше для добавления идентификатора к URI контента `UserDictionary` используется метод [withAppendedId\(\)](#).



# Получение данных от поставщика

В это разделе рассматривается порядок получения данных от поставщика на примере поставщика пользовательского словаря.

Для полной ясности в примерах кода, приведенных в этом разделе, методы [ContentResolver.query\(\)](#) вызываются в потоке пользовательского интерфейса.

В реальном коде запросы следует выполнять асинхронно в отдельном потоке.

Одним из способов реализовать это является использование класса [CursorLoader](#), который более подробно описан в статье [Загрузчики](#).

Чтобы получить данные из поставщика, выполните указанные ниже основные действия.

- Запросите у поставщика разрешение на чтение.
- Определите код, который отвечает за отправку запроса поставщику.

# Запрос разрешения на чтение

Чтобы ваше приложение могло получать данные от поставщика, приложению требуется получить от поставщика разрешение на чтение.

Это разрешение невозможно получить во время выполнения; вместо этого вам необходимо указать, что вам требуется такое разрешение, в манифесте приложения.

Для этого воспользуйтесь элементом [<uses-permission>](#) и укажите точное название разрешения, определенное поставщиком.

Указав этот элемент в манифесте, вы тем самым запрашиваете необходимое разрешение для вашего приложения.

Когда пользователи устанавливают ваше приложение, они косвенно получают разрешение по этому запросу.

# Разрешения на чтение

Чтобы узнать точное название разрешения на чтение в используемом поставщике, а также названия других используемых в нем разрешений на чтение, обратитесь к документации поставщика.

Дополнительные сведения о роли разрешений в получении доступа к поставщику представлены в разделе [Разрешения поставщика контента](#).

Поставщик пользовательского словаря задает разрешение `android.permission.READ_USER_DICTIONARY` в своем файле манифеста, поэтому приложению, которому требуется выполнить чтение данных из поставщика, необходимо запросить именно это разрешение.

# Создание запроса

Следующим этапом получения данных от поставщика является создание запроса. В следующем фрагменте кода задаются некоторые переменные для доступа к поставщику пользовательского словаря:

```
// "проекция" определяет столбцы, которые будут возвращены для каждой строки
String[] mProjection =
{
    UserDictionary.words._id, // Contract Class constant для имени столбца _id
    UserDictionary.words.WORD, // Contract Class constant для имени столбца
word
    UserDictionary. words.LOCALE // Contract Class constant for the locale column
name
};

// определяет строку, содержащую предложение выбора
String mSelectionClause = null;

// инициализирует массив, содержащий аргументы выбора
String[] mSelectionArgs = {""};
```

# Порядок использования метода [ContentResolver.query\(\)](#)

В следующем фрагменте кода демонстрируется порядок использования метода [ContentResolver.query\(\)](#) (в качестве примера выступает поставщик пользовательского словаря):

Клиентский запрос поставщика аналогичен SQL-запросу. В нем содержится набор столбцов, которые возвращаются, набор критериев выборки и порядок сортировки.

Набор столбцов, которые должен вернуть запрос, называется **проекцией** (переменная `mProjection`).

Выражение, которое задает строки для получения, состоит из предложения выбора и аргументов выбора.

Предложение выбора представляет собой сочетание логических выражений, имен столбцов и значений (переменная `mSelectionClause`). Если вместо значения указать подставляемый параметр `?`, метод запроса извлекает значение из массива аргументов выбора (переменная `mSelectionArgs`).

В следующем фрагменте кода, если пользователь не указал слово, то для предложения выбора задается значение `null`, а запрос возвращает все слова, имеющиеся в поставщике.

Если пользователь указал слово, то для предложения выбора задается значение `UserDictionary.Words.WORD + " = ?"`, а для первого элемента в массиве аргументов выбора задается введенное пользователем слово.

# ContentResolver.query()

```
/** определяет одноэлементный строковый массив, содержащий аргумент
    выбора.*/
String[] mSelectionArgs = {""};
// получает слово из пользовательского интерфейса
mSearchString = mSearchWord.getText().toString();
// не забудьте вставить код здесь, чтобы проверить наличие недопустимого
или вредоносного ввода.
// Если слово является пустой строкой, то получает все
if (TextUtils.isEmpty(mSearchString)) {
    // установка предложения selection в null вернет все слова
    mSelectionClause = null;
    mSelectionArgs[0] = "";
} else {
    // создает предложение selection, соответствующее слову, введенному
    пользователем.
    mSelectionClause = UserDictionary.Слова.Слово + "= ?";
    // Перемещает строку ввода пользователя в аргументы выбора.
    mSelectionArgs[0] = mSearchString;}
```

# ContentResolver.query()

```
// выполняет запрос к таблице и возвращает объект Курсора
mCursor = getContentResolver().query(
    UserDictionary. words.CONTENT_URI, // содержание URI слов таблицы
    mProjection,                       // столбцов, возвращаемых для каждой строки
    mSelectionClause // либо null, либо слово введенное пользователем
    mSelectionArgs,                    // либо пуст, либо содержит строки, введенные
    пользователем
    mSortOrder);                      // порядок сортировки для возвращенных строк
// некоторые операторы возвращают значение NULL, если произошла ошибка, другие
бросают исключение,
if (null == mCursor) {
    /**вставьте код здесь, чтобы обработать ошибку. Убедитесь, что вы не используете
    курсор**/
    // Если курсор пуст, поставщик не нашел совпадений
} else if (mCursor.getCount() < 1) {
    /**Вставьте здесь код, чтобы уведомить Пользователя о том, что поиск был
    неудачным. Это не обязательно ошибка. Вы можете предложить пользователю
    возможность вставить новую строку или повторно ввести поисковый запрос*.*/
} else {
    /**вставьте код сюда, чтобы что-то сделать с результатами
    */
}
```

## Это аналогично

```
SELECT _ID, word, locale FROM words WHERE word =<userinput> ORDER BY word ASC;
```

В этой инструкции SQL вместо констант класса-контракта используются фактические имена столбцов.

# Защита от ввода вредоносного кода

Если данные, которыми управляет поставщик контента, находятся в базе данных SQL, то включение в необработанные инструкции SQL внешних ненадежных данных может привести к атаке путем внедрения кода SQL.

Рассмотрим следующее предложение выбора:

```
// Создает предложение выбора путем объединения входных данных Пользователя со строкой имени столбца
```

```
mSelectionClause = "var =" + mUserInput;
```

Если вы используете это предложение, вы разрешаете пользователю связать вашу инструкцию SQL с вредоносным кодом SQL.

Например, пользователь может ввести **nothing; DROP TABLE \*;** для `mUserInput`, что приведет к созданию следующего предложения выбора: **var = nothing; DROP TABLE \*;**

Поскольку предложение выбора выполняется в потоке как инструкция SQL, это может привести к тому, что поставщик удалит все таблицы в соответствующей базе данных SQLite (если только поставщик не настроен на отслеживание попыток [внедрить вредоносный код SQL](#)).



# Защита от ввода вредоносного кода

Чтобы избежать этого, воспользуйтесь предложением выбора, в котором ? выступает в качестве подставляемого параметра, а **также отдельным массивом аргументов выбора**.

После этого ввод пользователя будет связан напрямую с запросом и не будет интерпретироваться как часть инструкции SQL.

Поскольку в этом случае введенный пользователем запрос не рассматривается как код SQL, то в него не удастся внедрить вредоносный код SQL.

Вместо объединения, которое следует включить в пользовательский ввод, используйте следующее предложение выбора:

```
// Создает предложение selection со сменной  
// строкой параметров mSelectionClause = " var = ?";  
Настройте массив аргументов выбора следующим образом:
```

```
// Определяет массив, содержащий строку аргументов выбора  
argumentsString[] selectionArgs ={" "};
```

Укажите значение для массива аргументов выбора:

```
// Устанавливает аргумент выбора на вход пользователя  
selectionArgs[0] = userInput;
```

Предложение выбора, в котором ? используется в качестве подстановочного параметра, и массив аргументов выбора представляют собой предпочтительный способ указания выбора, даже если поставщик не использует базу данных SQL.

# Отображение результатов запроса

Клиентский метод [ContentResolver.query\(\)](#) всегда возвращает объект [Cursor](#), содержащий столбцы, указанные в проекции запроса для строк, которые соответствуют критериям выборки в запросе.

Объект [Cursor](#) предоставляет прямой доступ на чтение содержащихся в нем строк и столбцов.

С помощью методов [Cursor](#) можно выполнить итерацию по строкам в результатах, определить тип данных для каждого столбца, получить данные из столбца, а также проверить другие свойства результатов.

Некоторые реализации объекта [Cursor](#) автоматически обновляют объект при изменении данных в поставщике или запускают выполнение методов в объекте-наблюдателе при изменении объекта [Cursor](#), либо выполняют и то, и другое.

Если строки, соответствующие критериям выборки, отсутствуют, поставщик возвращает объект [Cursor](#), в котором для метода [Cursor.getCount\(\)](#) указано значение «0» (пустой объект cursor).

# Работа с курсором

При возникновении внутренней ошибки результаты запроса зависят от определенного поставщика. Поставщик может вернуть null или выдать [Exception](#).

Поскольку [Cursor](#) представляет собой «список» строк, то наилучшим способом отобразить содержимое объекта [Cursor](#) будет связать его с [ListView](#) посредством [SimpleCursorAdapter](#).

Следующий фрагмент кода является продолжением предыдущего фрагмента.

Он создает объект [SimpleCursorAdapter](#), содержащий объект [Cursor](#), который был получен в запросе, а затем определяет этот объект в качестве адаптера для [ListView](#):

# Пример

```
// Определяет список столбцов для извлечения из курсора и загрузки в выходную строку
String[] mWordListColumns =
{   UserDictionary.words.WORD, // Contract константа класса, содержащая имя столбца word
    UserDictionary.words.LOCALE // Contract константа класса, содержащая имя столбца locale
};
// определяет список идентификаторов представления, которые будут получать столбцы
курсора для каждой строки
int[] mWordListItems = { R.id.dictWord, R.id.locale};
// создает новый SimpleCursorAdapter
mCursorAdapter = new SimpleCursorAdapter(
getApplicationContext(), // контекстный объект приложения
R.layout.wordlistrow, // Макет в XML для одной строки в ListView
mCursor, // результат запроса
mWordListColumns, // строковый массив имен столбцов в курсоре
mWordListItems, // целочисленный массив идентификаторов представлений в
строке layout
0); // флаги (обычно они не нужны)
// устанавливает адаптер для ListView
mWordList.setAdapter(mCursorAdapter);
```

**Примечание.** Чтобы вернуть [ListView](#) с объектом [Cursor](#), объект cursor должен содержать столбец с именем `_ID`. Поэтому показанный ранее запрос извлекает столбец `_ID` для таблицы `words`, даже если [ListView](#) не отображает ее. Данное ограничение также объясняет, почему в каждой таблице поставщика имеется столбец `_ID`

# Получение данных из результатов запроса

Вместо того, чтобы просто отобразить результаты запроса, вы можете использовать их для выполнения других задач.

Например, можно получить написание слов из пользовательского словаря, а затем выполнить их поиск в других поставщиках.

Для этого выполните итерацию по строкам в объекте [Cursor](#):

```
// Определите индекс столбца столбца с именем "word"
int index = mCursor.getColumnIndex(UserDictionary.WORD);
/** выполняется только в том случае, если курсор действителен. Поставщик
пользовательского словаря возвращает значение null, если * возникает внутренняя
ошибка. Другие поставщики могут выдавать исключение вместо возврата null.* /
if (mCursor != null) {
    /**переходит к следующей строке курсора. Перед первым перемещением курсора *
"указатель строки" равен -1, и если вы попытаетесь извлечь данные в этой позиции, то
получите исключение*.* /
    while (mCursor.moveToNext()) {
        //получает значение из столбца.
        newWord= mCursor.getString(index);
        // вставьте здесь код для обработки полученного слова.

        ...
    } // end of while loop
} else {
    //Insert code here to report an error if the cursor is null or the provider thrown an exception
}
```

Реализации объекта [Cursor](#) содержат несколько методов `get` для получения из объекта различных типов данных. Например, во фрагменте кода используется метод [getString\(\)](#). В них также имеется метод [getType\(\)](#), который возвращает значение, указывающее на тип данных в столбце.

# Разрешения поставщика контента

Приложение поставщика может задавать разрешения, которые требуются другим приложениям для доступа к данным в поставщике.

Такие разрешения гарантируют, что пользователь знает, к каким данным приложение будет пытаться получить доступ. На основе требований поставщика другие приложения запрашивают разрешения, которые требуются им для доступа к поставщику. Конечные пользователи видят запрошенные разрешения при установке приложения.

Если приложение поставщика не задает никаких разрешений, другие приложения не получают доступ к данным поставщика. Однако компонентам приложения поставщика всегда предоставлен полный доступ на чтение и запись, независимо от заданных разрешений.

Для получения данных из поставщика пользовательского словаря требуется разрешение **android.permission.READ\_USER\_DICTIONARY**.

В поставщике предусмотрено отдельное разрешение **android.permission.WRITE\_USER\_DICTIONARY** для вставки, обновления или удаления данных.

Чтобы получить разрешения, необходимые для доступа к поставщику, приложение запрашивает их с помощью элемента [<uses-permission>](#) в файле манифеста.

При установке менеджером пакетов Android приложения пользователю необходимо утвердить все разрешения, запрашиваемые приложением.

В случае утверждения всех разрешений менеджер пакетов продолжает установку; если же пользователь отклоняет их, менеджер пакетов отменяет установку.

Для запроса доступа на чтение данных в поставщике пользовательского словаря используется следующий элемент [<uses-permission>](#):

```
<uses-permission android: name=" android.разрешение.READ_USER_DICTIONARY">
```

Дополнительные сведения о влиянии разрешений на доступ к поставщику представлены в статье [Безопасность и разрешения](#).

# Вставка данных

Для вставки данных в поставщик вызовите метод [ContentResolver.insert\(\)](#). Этот метод вставляет новую строку в поставщик и возвращает URI контента для этой строки. В следующем фрагменте кода демонстрируется порядок вставки нового слова в поставщик пользовательского словаря:

```
// Defines a new Uri object that receives the result of the insertion
Uri mNewUri;

...
// Defines an object to contain the new values to insert
ContentValues mNewValues = new ContentValues();

/*
 * Sets the values of each column and inserts the word. The arguments to the "put"
 * method are "column name" and "value"
 */
mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "insert");
mNewValues.put(UserDictionary.Words.FREQUENCY, "100");

mNewUri = getContentResolver().insert(
    UserDictionary.Word.CONTENT_URI, // the user dictionary content URI
    mNewValues                       // the values to insert
);
```

# Комментарий

Данные для новой строки поступают в один объект [ContentValues](#), который аналогичен объекту `cursor` с одной строкой. Столбцы в этом объекте необязательно должны содержать данные такого же типа, и если вы вообще не собираетесь указывать значение, вы можете задать для столбца значение `null` с помощью метода [ContentValues.putNull\(\)](#).

Код в представленном фрагменте не добавляет столбец `_ID`, поскольку этот столбец сохраняется автоматически. Поставщик присваивает уникальное значение `_ID` каждой добавляемой строке. Обычно поставщики используют это значение в качестве основного ключа таблицы.

URI контента, возвращенный в элементе `newUri`, служит для идентификации новой добавленной строки в следующем формате:

```
content://user_dictionary/words/<id_value>
```

`<id_value>` — это содержимое столбца `_ID` для новой строки. Большинство поставщиков автоматически определяют эту форму URI контента, а затем выполняют запрошенную операцию с требуемой строкой.

Чтобы получить значение `_ID` из возвращенного объекта [Uri](#), вызовите метод [ContentUris.parseId\(\)](#).



# Обновление данных

Чтобы обновить строку, используйте объект [ContentValues](#) с обновленными значениями (точно так же, как вы это делаете при вставке) и критериями выборки (так же, как и с запросом). Используемый вами клиентский метод называется [ContentResolver.update\(\)](#).

Вам не нужно добавлять значения в объект [ContentValues](#) для обновляемых столбцов. Чтобы очистить содержимое столбца, задайте значение null.

Следующий фрагмент кода служит для изменения языка во всех строках, где в качестве языка указано en, на null. Возвращаемое значение представляет собой количество строк, которые были обновлены:

# Пример

```
// Defines an object to contain the updated values
ContentValues mUpdateValues = new ContentValues();

// Defines selection criteria for the rows you want to update
String mSelectionClause = UserDictionary.Words.LOCALE + "LIKE ?";
String[] mSelectionArgs = {"en_%"};

// Defines a variable to contain the number of updated rows
int mRowsUpdated = 0;
...

/* * Sets the updated value and updates the selected words. */
mUpdateValues.putNull(UserDictionary.Words.LOCALE);

mRowsUpdated = getContentResolver().update(
    UserDictionary.Words.CONTENT_URI, // the user dictionary content URI
    mUpdateValues                    // the columns to update
    mSelectionClause                  // the column to select on
    mSelectionArgs                    // the value to compare to
);
```

# Удаление данных

Удаление данных аналогично получению данных строки: необходимо указать критерии выборки для строк, которые требуется удалить, после чего клиентский метод возвратит количество удаленных строк.

Ниже представлен фрагмент кода для удаления строк с идентификатором appid user.

Метод возвращает количество удаленных строк.

```
// Defines selection criteria for the rows you want to delete
String mSelectionClause = UserDictionary.Words.APP_ID + " LIKE ?";
String[] mSelectionArgs = {"user"};

// Defines a variable to contain the number of rows deleted
int mRowsDeleted = 0;

...
// Deletes the words that match the selection criteria
mRowsDeleted = getContentResolver().delete(
    UserDictionary.Words.CONTENT_URI, // the user dictionary content URI
    mSelectionClause // the column to select on
    mSelectionArgs // the value to compare to
);
```

Также следует проверить пользовательский ввод при вызове метода [ContentResolver.delete\(\)](#). Дополнительные сведения об этом представлены в разделе [Защита от ввода вредоносного кода](#).

# Типы поставщиков данных

Поставщики контента могут предоставлять различные тип данных. Поставщик пользовательского словаря предоставляет только текст, но также может предоставлять следующие форматы:

целое число;

длинное целое число (long);

число с плавающей запятой;

длинное число с плавающей запятой (double).

Другим типом данных, предлагаемых поставщиком, является большой двоичный объект (BLOB), реализованный как 64-разрядный массив.

Чтобы просмотреть доступные типы данных, обратитесь к методам get класса [Cursor](#).

Тип данных для каждого столбца в поставщике обычно указывается в документации к поставщику.

Типы данных для поставщика пользовательского словаря указаны в справочной документации для класса-контракта [UserDictionary.Words](#) (дополнительные сведения о классах-контрактах представлены в разделе [Классы-контракты](#)).

Также определить тип данных можно путем вызова метода [Cursor.getType\(\)](#).

Поставщики также хранят информацию о типе данных MIME для каждого определяемого ими URI контента.

Эту информацию можно использовать для определения того, может ли ваше приложение обрабатывать предлагаемые поставщиком данные, а также для выбора типа обработки на основе типа MIME.

Информация о типе MIME обычно требуется при работе с поставщиком, который содержит сложные структуры данных или файлы.

Например, в таблице [ContactsContract.Data](#) в поставщике контактов используются типы MIME для отметки типа данных контакта, которые хранятся в каждой строке. Чтобы получить тип MIME, соответствующий URI контента, вызовите метод [ContentResolver.getType\(\)](#).

Синтаксис стандартных и настраиваемых типов MIME описан в [справке по типам MIME](#)

# Литература

- <https://developer.android.com/guide/topics/providers/content-provider-basics>