**16.1 Введение**

Будем двигаться дальше в рассмотрении вопросов связанных с разработкой приложений для смартфонов.

Современное *программирование* трудно представить без использования баз данных, рано или поздно в процессе развития приложения появляется осознание необходимости долговременного хранения и обработки структурированной информации. Данная лекция посвящена рассмотрению вопросов, связанных с использованием баз данных SQLite в приложениях, разрабатываемых под *Android*. *Базы данных* SQLite являются основой построения рабочей и функциональной программы, в которой необходимо работать с большими объемами структурированной информации.

Далее в лекции перейдем к рассмотрению таких интересных тем, как создание графических изображений и анимации, а также работа с этими элементами. Платформа *Android* предоставляет разнообразные способы для добавления в приложения и использования графики и анимации.

Очень часто мобильные устройства помогают "скоротать время" в очередях, в ожидании транспорта и многих других ситуациях, часто возникающих в современной жизни. Проще всего в такие моменты занять себя несложной игрой, в связи с этим тема разработки игр для мобильных устройств стала довольно популярна в последнее время. Разумеется, разработка игр дело серьезное, но даже отдельному разработчику *по* силам создать игру, способную увлечь пользователя. В данной теме рассмотрим основные принципы создания игр для смартфонов, в лабораторной работе рассмотрим процесс создания несложной игры.

**16.2 Основы работы с базами данных, SQLite**

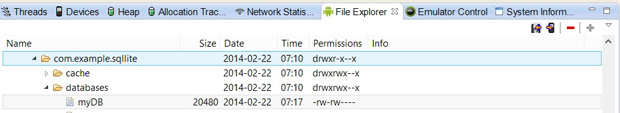
SQLite - небольшая и при этом мощная система управления базами данных. Эта система создана в 2000 году, ее разработчик доктор Ричард Хипп (Dr. Richard Hipp). В настоящее время является одной из самых распространенных *SQL*-систем управления базами данных в мире. Можно выделить несколько причин такой популярности SQLite: она бесплатная; она маленькая, примерно 150 Кбайт; не требует установки и администрирования. Подробнее см. [http://www.sqlite.org](http://www.sqlite.org/).

*База данных* SQLite - это обычный *файл*, его можно перемещать и копировать на другую систему (например, с телефона на рабочий *компьютер*) и она будет отлично работать. *Android* хранит *файл* *базы данных* приложения в папке (см. [рис. 16.1](http://www.intuit.ru/studies/courses/12643/1191/lecture/22001?page=1#image.16.1).):

data/data/packagename/databases/,

где packagename - *имя пакета*, в котором расположено *приложение*.

Для доступа к этому файлу необходимо запускать команды *SQL*, *Android* с помощью вспомогательных классов и удобных методов скрывает часть деталей, но все таки необходимо иметь хотя бы минимальные знания об *SQL*, чтобы пользоваться этими инструментами.

[](http://www.intuit.ru/EDI/14_06_16_2/1465856498-31733/tutorial/1258/objects/16/files/16_01.jpg)

[увеличить изображение](http://www.intuit.ru/EDI/14_06_16_2/1465856498-31733/tutorial/1258/objects/16/files/16_01.jpg)  
**Рис. 16.1.**Расположение файла базы данных SQLite

Обращения к базе данных *SQL* выполняются посредством запросов, существует три основных вида *SQL* запросов: *DDL*, *Modification* и Query.

* **DDL запросы.** Такие запросы используются для создания таблиц. Каждая таблица характеризуется именем и описанием столбцов, которое содержит имя столбца и тип данных. В файле базы данных может быть несколько таблиц.

Пример запроса для создания таблицы:

create Table\_Name (

\_id integer primary key autoincrement,

field\_name\_1 text,

field\_name\_2 text);

Первый столбец обозначен, как primary key (первичный ключ), т. е. уникальное число, которое однозначно идентифицирует строку. Слово autoincrement указывает, что база данных будет автоматически увеличивать значение ключа при добавлении каждой записи, что и обеспечивает его уникальность. Существует договоренность первый столбец всегда называть \_id, это не жесткое требование SQLite, однако может понадобиться при использовании контент-провайдера в Android.

Стоит иметь в виду, что в SQLite, в отличие от многих других баз данных, типы данных столбцов являются лишь подсказкой, т. е. не вызовет никаких нареканий попытка записать строку в столбец, предназначенный для хранения целых чисел или наоборот. Этот факт можно рассматривать, как особенность базы данных, а не как ошибку, на это обращают внимание авторы SQLite.

* **Modification запросы.** Такие запросы используются для добавления, изменения или удаления записей.

Пример запроса на добавление строки:

insert into Table\_Name values(null, value1, value2);

В этом случае значения разместятся в соответствующие столбцы таблицы, первое значение задается для поля \_id и равно null, т. к. SQLite вычисляет значение этого поля самостоятельно.

При добавлении можно указывать столбцы, в которые будут размещаться значения, остальные столбцы заполнятся значениями по умолчанию, в этом случае можно добавлять элементы в измененном порядке. Пример такого запроса:

insert into Table\_Name(field\_name\_2, field\_name\_1)

values(value2, value1);

В этом случае добавляются значения только в поля field\_name\_1 и field\_name\_2, причем изменен порядок следования полей, а вместе с этим и порядок следования значений, иногда это бывает удобно.

Примеры запросов на изменение строки:

update Table\_Name set Field\_Name\_1 = value;

поменяет значение столбца Field\_Name\_1 на value во всей таблице;

update Table\_Name set Field\_Name\_1 = value where \_id = smth;

поменяет значение столбца Field\_Name\_1 только в той строке, \_id которой равен smth.

Примеры запросов на удаление строк:

delete from Table\_Name;

delete from Table\_Name where Field\_Name\_1 = smth;

первый запрос удаляет все строки таблицы, второй - только те строки, в которых столбец Field\_Name\_1 имеет значение smth.

* **Query запросы.** Такие запросы позволяют получать выборки из таблицы по различным критериям. Пример запроса:
* select \* from Table\_Name where (\_id = smth);
* select Field\_Name\_1, Field\_Name\_2 from Table\_Name
* Field\_Name\_1 = smth);

Первый *запрос* выводит строку с \_id равным smth, второй - выводит два элемента Field\_Name\_1 и Field\_Name\_2 строк, в которыхField\_Name\_1 равен smth.

Вернемся к рассмотрению вопросов, связанных с использованием *базы данных* SQLite в приложениях под *Android*. Любая *база данных*, созданная в приложении доступна любому классу приложения, но недоступна из вне. Чтобы открыть *доступ* к базе данных другим приложениям необходимо использовать контент-провайдеры (*Content* Providers).

Для создания и обновления *базы данных* в *Android* предусмотрен *класс* SQLiteOpenHelper. При разработке приложения, работающего с базами данных, необходимо создать *класс*-наследник от SQLiteOpenHelper, в котором обязательно реализовать методы:

|  |  |
| --- | --- |
| onCreate() | - вызывается при первом создании базы данных; |
| onUpgrade() | - вызывается, когда необходимо обновить базу данных. |

*По* желанию можно реализовать метод:

|  |  |
| --- | --- |
| onOpen() | - вызывается при открытии базы данных. |

В этом же классе имеет смысл объявить строковые *константы*, в которых определить названия таблиц и столбцов. Полученный *класс*позаботится об открытии *базы данных*, если она существует, или о создании ее в противном случае, а так же об обновлении *базы данных* в случае необходимости.

В *Android* предусмотрен *класс* для работы с базой данных SQLite напрямую, этот *класс* называется SQLiteDatabase и содержит методы:

|  |  |
| --- | --- |
| openDatabase() | - позволяет открыть базу данных; |
| update() | - позволяет обновить строки таблицы базы данных; |
| insert() | - позволяет добавлять строки в таблицу базы данных; |
| delete() | - позволяет удалять строки из таблицы базы данных; |
| query() | - позволяет составлять запросы к базе данных; |
| execSQL() | - позволяет выполнять запросы к базе данных. |

Для добавления новых строк в таблицу используется *класс* ContentValues, каждый *объект* этого класса представляет собой одну строку таблицы и выглядит как ассоциативный *массив* с именами столбцов и значениями, которые им соответствуют.

Для получения результатов запросов к базе данных используется *класс* Cursor, объекты этого класса ссылаются на результирующий набор данных, позволяют управлять текущей позицией в возвращаемом при запросе наборе данных.

Для предоставления доступа к данным для других приложений можно использовать контент-провайдеры (ContentProvider). Любая *информация*, управляемая контент-провайдером адресуется посредством *URI*:

content://authority/path/id

где:

|  |  |
| --- | --- |
| content:// | - стандартный требуемый префикс; |
| authority | - имя провайдера, рекомендуется использовать полное квалификационное имя пакета для избежания конфликта имен; |
| path | - виртуальная папка внутри провайдера, которая определяет вид запрашиваемых данных; |
| id | - первичный ключ отдельной запрошенной записи, для запроса всех записей определенного типа этот параметр не указывается. |

Контент-провайдеры поддерживают стандартный *синтаксис* запросов для чтения, изменения, вставки и удаления данных.

Подробнее работу с SQLite базами данных в приложениях под *Android* рассмотрим в первой части лабораторной работы в этой теме.

**16.3 Анимация**

*Android* предоставляет мощные *API* для анимации элементов пользовательского интерфейса и построения 2D и *3D* изображений.

Платформа *Android* предоставляет две системы анимации: *анимация* свойств, появившаяся в *Android* 3.0, и *анимация* компонентов пользовательского интерфейса (наследников класса View). Рассмотрим подробнее обе эти системы.

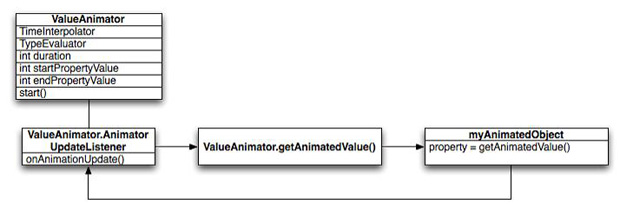
**Анимация свойств (Property Animation).** Система анимации свойств позволяет определить анимацию для изменения любого свойства объекта, независимо от того изображается оно на экране или нет. Используя эту систему, можно задать следующие характеристики анимации:

* **Продолжительность** предполагает задание длительности временного промежутка выполнения анимации, по умолчанию это значение равно 300 мс.
* **Временная интерполяция** предполагает вычисление значения свойства в каждый момент времени, как функции от промежутка времени, прошедшего с начала анимации.
* **Количество повторов и поведение** определяет необходимость повторения анимации при достижении конца заданного временного промежутка, а также количество повторов в случае необходимости. Эта же характеристика позволяет задать возможность воспроизведения в обратном порядке, если эта возможность выбрана, то анимация прокручивается вперед-назад заданное число раз.
* **Группа анимаций** позволяет организовать анимации в некоторое множество и задать режим исполнения: одновременно, последовательно непрерывно или с некоторыми задержками.
* **Частота обновления кадров** определяет, как часто будет происходить смена кадров анимации. По умолчанию обновление происходит каждые 10 мс, однако скорость, с которой приложение сможет обновлять кадры, в конечном итоге, зависит от загруженности системы.

Большая часть *API* системы анимации свойств находится в пакете **android.animation**. Также можно использовать блоки интерполяции, определенные в пакете **android.view.animation**.

*Класс* Animator предоставляет базовую структуру для создания анимации. Напрямую этот *класс* обычно не используется, так как обеспечивает минимальную функциональность, поэтому чаще всего используются классы-наследники, расширяющие возможности классаAnimator. Рассмотрим основные классы, используемые для создания анимации свойств.

* ValueAnimator (потомок класса Animator). Этот класс является главным обработчиком распределения времени для анимации свойств, а также рассчитывает значения свойства, предназначенного для анимации. Он обеспечивает всю основную функциональность: рассчитывает значения анимации и содержит распределенные во времени детали каждой анимации; содержит информацию о необходимости повторений анимации; содержит слушателей, получающих уведомления о событиях обновления; предоставляет возможность задавать пользовательские типы для вычисления. В процессе анимации свойств можно выделить две части: вычисление значения свойства, для которого определяется анимация, и присвоение полученного значения соответствующему полю объекта.ValueAnimator не выполняет вторую часть, поэтому необходимо следить за обновлениями значений, вычисляемых в классеValueAnimator, и изменять объекты, подверженные анимации. Наглядно рассмотренные части анимации с использованием классаValueAnimator представлены на [рис. 16.2](http://www.intuit.ru/studies/courses/12643/1191/lecture/22001?page=2#image.16.2).

[](http://www.intuit.ru/EDI/14_06_16_2/1465856498-31733/tutorial/1258/objects/16/files/16_02.jpg)

[увеличить изображение](http://www.intuit.ru/EDI/14_06_16_2/1465856498-31733/tutorial/1258/objects/16/files/16_02.jpg)  
**Рис. 16.2.**Процесс анимации свойств с использованием класса ValueAnimator

* AnimatorSet (потомок класса Animator). Предоставляет механизмы группировки анимаций, таким образом, что они выполняются некоторым образом относительно друг друга. Можно определять выполнение анимаций одновременно, последовательно и с временными задержками.

**Классы-вычислители** определяют как вычислять значения заданных свойств. Они получают: данные о *распределение времени*, предоставляемые классом Animator, начальное и конечное значения свойства, после чего на основе этих данных вычисляют значения свойства, для которого выполняется *анимация*. В системе анимации свойств существуют следующие вычислители:

* IntEvaluator для вычисления целочисленных значений свойств;
* FloatEvaluator для вычисления вещественных значений свойств;
* ArgbEvaluator для вычисления значений цвета в шестнадцатеричном представлении;
* TypeEvaluator - интерфейс, позволяющий создавать собственных вычислителей.

**Интерполяторы** определяют с помощью каких функций от времени, вычисляются значения свойств, для которых задается *анимация*. Интерполяторы определены в пакете *android*.view.*animation*. Если ни один из существующих интерполяторов не подходит, можно создать собственный, реализовав *интерфейс* **TimeInterpolator**.

Подробнее с системой анимации свойств можно познакомиться *по* ссылке: <http://developer.android.com/guide/topics/graphics/prop-animation.html>.

**Анимация компонентов пользовательского интерфейса.** Эта система может быть использована для реализации анимации преобразований над наследниками класса View. Для расчета анимации преобразований используется следующая *информация*: начальная точка, конечная точка, размер, поворот и другие общие аспекты анимации. *Анимация* преобразований может выполнять серии простых изменений содержимого экземпляра класса View. Например, для текстового поля можно перемещать, вращать, растягивать, сживать текст, если определено фоновое изображение, оно должно изменяться вместе с текстом. Пакет **android.view.animation** предоставляет все классы, необходимые для реализации анимации преобразований.

Для задания последовательности инструкций анимации преобразований можно использовать или *XML*, или *Android* код. Более предпочтительным является *определение* анимации в *XML* файлах, располагаться эти файлы должны в папке **res/anim/** проекта. *XML* *файл*должен иметь единственный *корневой элемент*, это может быть любой из отдельных элементов: <alpha>, <scale>, <translate>,<rotate>, интерполятор, или же элемент <set>, который содержит группы этих элементов, в том числе может содержать другие элементы<set>. *По* умолчанию инструкции анимации выполняются одновременно, чтобы задать последовательное *исполнение* необходимо определить*атрибут* startOffset.

Подробнее с системой анимации преобразований можно познакомиться *по* ссылке: <http://developer.android.com/guide/topics/graphics/view-animation.html>.

Дополнительно к рассмотренным системам анимации может использоваться, кадровая *анимация*, которая реализуется быстрой сменой кадров, каждый *кадр* является графическим ресурсом и располагается в папке **res/drawable/** проекта.

Подробнее с кадровой анимацией можно познакомиться *по* ссылке: <http://developer.android.com/guide/topics/graphics/drawable-animation.html>.

**16.4 2D и 3D графика**

При разработке приложения важно четко понимать требования к графике в этом приложении. Для разных графических задач необходимы разные техники их решения. Далее в лекции рассмотрим несколько способов изображения графических объектов в *Android*.

**Холсты и графические объекты.** Платформа *Android* предоставляет *API* для изображения 2D графики, который позволяет изображать на холсте свои графические объекты или изменять существующие. Для отображения 2D графики существуют два пути:

1. Изобразить графику или анимацию в элементе пользовательского интерфейса. В этом случае графика управляется процессом отображения иерархии элементов интерфейса. Подходит, когда необходимо отобразить простую графику, не требующую динамических изменений.
2. Изображать графику напрямую на холсте (класс Canvas). В этом случае необходимо позаботиться о вызове метода onDraw(), передавая его в класс Canvas, или же о вызове одного из draw...() методов класса Canvas (например, drawPicture()). Действуя таким образом, можно управлять анимацией. Этот путь подходит, когда необходимо постоянно перерисовывать окно приложения, например, для видео игр.

**Аппаратное ускорение.** Начиная с *Android* 3.0 (*API* уровень 11), конвейер изображения 2D графики в *Android* поддерживает аппаратное ускорение. Это означает, что все *операции* рисования на холсте исполняются с использованием *GPU*. В связи с увеличением требований к ресурсам *приложение* будет потреблять больше *RAM*. Аппаратное ускорение доступно *по* умолчанию, если целевой уровень *API* больше или равен 14, но может быть включено явно. Если в приложении используются только стандартные представления и *графика*, включение аппаратного ускорения не должно привести к каким-либо нежелательным графическим эффектам. Однако из-за того, что аппаратное ускорение поддерживается не всеми операциями 2D графики, его включение может нарушать некоторые пользовательские изображения или вызовы рисования. Проблемы обычно проявляются в невидимости некоторых элементов, появлении исключений или неверно изображенных пикселях. Чтобы исправить это, *Android* позволяет включать или выключать аппаратное ускорение на разных уровнях: уровень приложения, уровень активности, уровень окна, уровень элемента интерфейса.

**OpenGL.** *Android* поддерживает высокопроизводительную 2D и *3D* графику с использованием открытой графической библиотеки OpenGL, точнее OpenGL ES *API*. Библиотека OpenGL является кросс-платформенным *API*, который определяет стандартный программный *интерфейс* для аппаратного обеспечения, занимающегося обработкой *3D* графики. OpenGL ES является разновидностью OpenGL, предназначенной для встроенных устройств. *Android* поддерживает несколько версий OpenGL ES *API*:

* OpenGL ES 1.0 и 1.1 поддерживается Android 1.0 и выше;
* OpenGL ES 2.0 поддерживается Android 2.2 (API уровень 8) и выше;
* OpenGL ES 3.0 поддерживается Android 4.3 (API уровень 18) и выше.

*Поддержка* OpenGL ES 3.0 на реальном устройстве требует реализации графического конвейера, предоставленной производителем. Поэтому устройство с *Android* 4.3 и выше может не поддерживать OpenGL ES 3.0.

Подробнее с графикой в *Android* можно познакомиться *по* ссылкам:

|  |
| --- |
| <http://developer.android.com/guide/topics/graphics/2d-graphics.html>; |
| <http://developer.android.com/guide/topics/graphics/opengl.html>; |
| <http://developer.android.com/guide/topics/graphics/hardware-accel.html>. |

**16.5 Основные принципы разработки игровых приложений для смартфонов**

Разработка игр дело обычно благодарное т. к. в игры люди играли, играют и будут играть. Даже если результат работы не принесет особой прибыли, в любом случае он способен доставить радость детям и друзьям, да и о себе забывать не стоит. При этом сам процесс разработки способен серьезно повысить уровень мастерства особенно начинающего разработчика.

Если возникло острое желание создать именно игровое *приложение*, необходимо иметь в виду некоторые особенности: практически любая *игра*предполагает наличие сюжета, игры обычно отличаются эффектным графическим оформлением и обеспечивают определенный игровой процесс (геймплей). И эти моменты стоит хорошо продумать прежде, чем начинать *программирование*.

Сюжет игры состоит из последовательности событий. Необходимость сюжета больше всего зависит от жанра игры: в некоторых жанрах можно обойтись совсем без сюжета. Не стоит, как недооценивать, так и переоценивать важность сюжета, т. к. он является лишь одной из составляющих успеха игры. Решение о том нужен или ненужен сюжет в игре, если нужен, то в какой мере и каким образом он будет выстраиваться, необходимо принимать взвешенно и до начала разработки.

При выборе способа графического оформления игры стоит иметь в виду, что использование *3D* графики серьезно усложнит процесс разработки, даже несложная *3D* *игра* отнимет очень много времени. В большинстве игр для мобильных устройств достаточно 2D графики, особенно в случае начинающего разработчика или команды таковых.

Следует учитывать ограниченные возможности мобильных устройств: сравнительно невысокая вычислительная *мощность*; ограниченный объем оперативной и дисковой памяти; небольшой размер и невысокое разрешение экрана; возможные проблемы, связанные с организацией передачи данных; ограниченный заряд аккумуляторных батарей.

Процесс разработки простого игрового приложения для смартфонов рассмотрен во второй части лабораторной работы к данной теме.