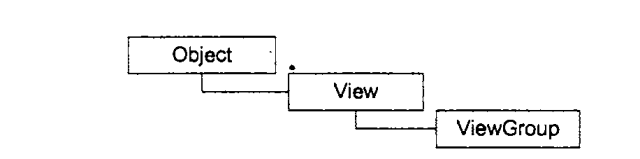
**Layouts (компоновки). Теория**

Переходим к изучению [Layouts](http://developer.android.com/guide/topics/ui/overview.html" \t "_blank). Или как часто переводят разметка, компоновка или представление элементов пользовательского интерфейса.  
  
Все элементы пользовательского интерфейса в приложениях Андроид создаются с использованием объектов [View](http://developer.android.com/reference/android/view/View.html" \t "_blank)и [ViewGroup](http://developer.android.com/reference/android/view/ViewGroup.html" \t "_blank).  
  
Класс View является базовым классом для ViewGroup и состоит из коллекции объектов View. Есть множество типов представлений и групп представлений, каждый из которых является потомком класса View.

**Иерархия классов View и ViewGropu**

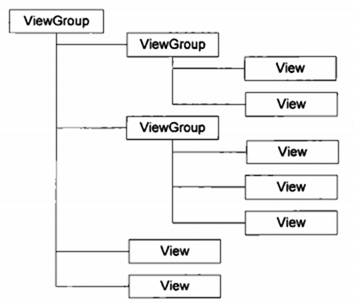
[](http://lh3.ggpht.com/-aATKyPx-yVA/VHhtsDmSAaI/AAAAAAAAFc8/ZSAmkr_cDEA/s1600-h/L0001%5B4%5D.png)

**Объекты View**— основные модули отображения интерфейса пользователя на платформе Android. Класс View служит ядром для подклассов, называемых виджетами, которые предлагают полностью реализованные объекты пользовательского интерфейса подобно текстовым полям, кнопкам и т. д. Объект View— структура данных, свойства которой сохраняют параметры разметки и содержание для определенной прямоугольной области экрана.

***Класс ViewGroup*** служит ядром для подклассов, называемых разметками (layouts), которые формируют расположение элементов пользовательского интерфейса на форме, используя различные виды архитектуры разметки — фреймовый, линейный, табличный и относительный.

На платформе Android необходимо определить пользовательский интерфейс для каждой Активности, используя иерархии узлов View и viewGroup. Это дерево иерархии может быть и простым, и сложным — в зависимости от требований к графическому интерфейсу приложения.

**Пример дерева Layouts** (представлений, разметок или компоновки)

[](http://lh4.ggpht.com/-_q-1q84Xe4E/VHhts-NUaII/AAAAAAAAFdI/emaS9qxzSGE/s1600-h/L0002%5B2%5D.png)

При запуске программы система Android получает ссылку на корневой узел дерева и использует ее для рисования графического интерфейса на экране мобильного устройства. Система также анализирует элементы дерева от вершины дерева иерархии, прорисовывая дочерние объекты view и ViewGroup и добавляя их родительским элементам.

При запуске Активности система должна получить ссылку на корневой узел дерева разметки, который будет использоваться для прорисовки графического интерфейса на экране мобильного устройства. Для этого в методе**onCreate()** необходимо вызвать метод **setСontentView()**, передав ему в качестве параметра ссылку на ресурс разметки в следующем виде:

**R.layout.layout\_file\_name**  
  
Например, если ваша XMLразметка сохранена как main.xml, вы загружаете разметку для Activity примерно так:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Override  public void onCreate(Bundle savedlnstanceState){   super.onCreate(savedlnstanceState);   setContentView(R.layout.main);  } |

Прорисовка начинается с корневого узла дерева разметки. Затем последовательно прорисовываются дочерние представления дерева разметки. Это означает, что родители будут прорисовываться раньше, чем их дочерние представления, — т. е. по окончании процесса прорисовки родители будут находиться на заднем плане по отношению к дочерним узлам.  
  
Разметка, как было сказано, — это архитектура расположения элементов интерфейса пользователя для конкретного окна, представляющего Активность. Она определяет структуру расположения элементов в окне и содержит все элементы, которые предоставляются пользователю программы. Разметку можно объявлять двумя способами:

* объявить элементы пользовательского интерфейса в XML. Android обеспечивает прямой XMLсловарь, который соответствует классам View и ViewGroup;
* создать разметку для окна в коде программы во время выполнения — инициализировать объекты Layout и дочерние объекты View, ViewGroup и управлять их свойствами программно.

Android позволяет использовать каждый из этих методов в отдельности или оба сразу для объявления и управления пользовательским интерфейсом в приложении. Например, можно объявить заданные по умолчанию разметки вашего приложения в XML, включая экранные элементы, которые появятся в них, и их свойства, а затем добавить код в вашем приложении, который во время выполнения изменит состояние объектов на экране, включая объявленные в XML.  
  
ADT-плагин для Eclipse предлагает удобный инструментарий — визуальный редактор разметки Layout Editor, который применяется для создания и предварительного просмотра создаваемых файлов разметки, которые находятся в каталоге res/layout/ проекта.  
  
Самый общий способ определять разметку и создавать иерархию представления— в XML файле разметки. XML предлагает удобную структуру для разметки, похожую на HTML разметку WEB страницы.  
  
Преимущество объявления пользовательского интерфейса в XMLфайле состоит в том, что это дает возможность отделить представление приложения от программного кода, который управляет поведением приложения. Ваше описание пользовательского интерфейса является внешним по отношению к программному коду, что означает, что вы можете изменять пользовательский интерфейс в файле разметки без необходимости изменения вашего программного кода.  
  
Например, вы можете создавать XML разметки для различных ориентаций экрана мобильного устройства (portrait, landscape), размеров экрана и языков интерфейса. Дополнительно, объявление разметки в XML файле облегчает возможность по визуализации структуры вашего пользовательского интерфейса, что упрощает отладку приложения.  
  
Каждый элемент в XML является объектом View или ViewGroup (или его потомком). Объекты view— листья в дереве, объекты ViewGroup— ветви. Вы можете также создавать объекты View и viewGroups в Java-коде, используя метод addview (View), чтобы динамически вставлять новые объекты view и ViewGroup в существующую разметку.  
  
Используя различные виды групп представлений, можно структурировать дочерние представления и группы представлений многими способами в зависимости от требований к графическому интерфейсу приложения. Некоторые стандартные группы представлений, предлагаемые Android (называющиеся разметками), включают LinearLayout, RelativeLayout, TableLayout и др. (будут подробно описаны позднее). Каждый из этих типов разметки предлагает уникальный набор параметров, которые используются, чтобы определить позиции дочерних представлений и структуру разметки на экране.  
Объявление в XML  
  
Используя XML словарь Android, можно быстро проектировать пользовательский интерфейс разметки и экранные элементы, которые он содержит, тем же самым способом, которым вы создаете web страницы в HTML — с рядом вложенных элементов.  
  
Каждый файл разметки должен содержать только один корневой элемент, который должен быть объектом View или ViewGroup. Как только вы определили корневой элемент, вы можете добавить дополнительные объекты разметки или виджеты как дочерние элементы, чтобы постепенно формировать иерархию элементов, которую определяет создаваемая разметка.  
  
Самый простой способ объяснять эту концепцию состоит в том, чтобы показать образец.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      android:orientation="vertical" >        <TextView          android:id="@+id/TextViewOl"          android:layout\_width="fill\_parent"          android:layout\_he=""          ight="wrap\_content"          android:text="@string/hello" >      </TextView>    </LinearLayout> |

Общая структура XML файла разметки проста: это — дерево XML элементов, где каждый узел представляет имя класса View (в нашем примере только один элемент View— TextView). Вы можете использовать имя любого класса, производного от класса View, как элемент в XML разметках, включая ваши собственные классы. Эта структура позволяет быстро создавать пользовательский интерфейс, используя более простую структуру и синтаксис, чем при создании разметки в программном коде.  
  
В вышеупомянутом XML примере есть корневой элемент **<LinearLayout>** и только один дочерний элемент View. TextView (текстовое поле), который имеет атрибуты. Описание некоторых из этих атрибутов приведем ниже:

* **xmlns:android** – декларация пространства имен XML, которая сообщает среде Android, что вы ссылаетесь на общие атрибуты, определенные в пространстве имен Android. В каждом файле разметки у корневого элемента должен быть этот атрибут со значением "<http://schemas.android.com/apk/res/android>"
* **android:layout\_width** – этот атрибут определяет, сколько из доступной ширины на экране должен использовать этот объект View (или View Group). В нашем случае он — единственный объект, таким образом, можно растянуть его на весь экран, которому в данном случае соответствует значение mutch\_parent
* **android:layout\_height** – аналогично android:layout\_width, за исключением того, что он ссылается на доступную высоту экрана
* **android:text** – устанавливает текст, который должен отобразить Textview. В этом примере используется строковый ресурс вместо жестко закодированного строкового значения. Строка hello определена в файле res/values/strings.xml. Это рекомендуемая практика для использования строковых ресурсов в ваших приложениях, потому что она делает локализацию приложения на другие языки более простой, без потребности кодировать напрямую изменения в файле разметки

Каждый объект View и ViewGroup поддерживает свое собственное разнообразие XML атрибутов. Некоторые атрибуты являются определенными только в объекте View (например, Textview поддерживает атрибут textSize), но эти атрибуты могут также наследоваться любыми объектами View, которые расширяют этот класс. Некоторые атрибуты являются общими ко всем объектам View, потому что они унаследованы от корневого класса View (подобно атрибутам id, layout\_width, layout\_height).  
  
В общем, XML словарь элементов пользовательского интерфейса близок к структуре классов и методов этих элементов, где имя элемента соответствует имени класса, а атрибуты элемента — методам этого класса. Фактически, соответствие является часто настолько точным, что легко предположить без обращения к документации Android, какой XML атрибут передает метод класса или, наоборот, какой метод класса соответствует конкретному XML элементу.  
  
Соответствие XML атрибутов и методов в классах представлений

|  |  |
| --- | --- |
| **Имя атрибута XML** | **Соответствующий метод в классе Java** |
| android:gravity | setGravity(int) |
| android: height | setHeight(int) |
| android:text | setText(CharSequence) |
| android:textColor | setTextColor(ColorStateList) |
| android:textSize | setrextSize (float) |
| android:width | setWidth(int) |
| android:textColorHighlight | setHighlightColor(int) |

Любой объект View в коде программы можно связать с объектом из файла разметки, однозначно определив элемент пользовательского интерфейса в пределах дерева. Когда приложение откомпилировано, на этот идентификатор ссылаются как на целое число, но идентификатор представления обычно обозначается в XML файле разметки как строка в атрибуте id. Синтаксис для идентификатора элемента в XMLтеге следующий:  
  
Символ **@** в начале строки указывает, что синтаксический анализатор XML должен проанализировать и развернуть остальную часть строки идентификатора и определить это выражение как ресурс идентификатора. Символ **+**означает, что это — новое имя ресурса, которое должно быть создано и добавлено к нашим ресурсам в файл R.java, который среда Android автоматически генерирует для проекта, как показано в листинге.  
  
Если в программном коде мы не работаем с некоторыми элементами пользовательского интерфейса, создавать идентификаторы для них необязательно, однако определение идентификаторов для объектов представления важно при создании RelativeLayout (относительной разметки). В RelativeLayout расположение представлений может определяться относительно другого представления, на которое ссылаются через его уникальный идентификатор:  
  
**android:id="@+id/TextViewOl"**  
  
Требование к уникальности идентификаторов не распространяется на все дерево представлений, но они должны быть уникальны в пределах части дерева (которая часто может быть и полным деревом представлений, так что лучше создавать полностью уникальные идентификаторы, если это возможно).  
  
Существуют следующие стандартные типы разметок, которые вы можете использовать в создаваемых приложениях:

* FrameLayout;
* LinearLayout;
* TableLayout;
* RelativeLayout;
* GridLayout.

Подобно всем разметкам, все они являются подклассами viewGroup и наследуют свойства, определенные в классе View.  
  
Что самое интересное весь этот материал взят от [сюда](http://androidfanclub.ru/programming/%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9-%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81-%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F), ну кроме [второго абзаца](http://developer.android.com/guide/topics/ui/overview.html) с верху который я сам перевел с альма-матер. Кто у кого стянул текст не понятно. То ли Голощапов, то ли этот сайт. Но на сайте инфы больше.