[Как создать custom View для Android Ч2.: Padding. Как нарисовать свой собственный контент](http://codeandlife.ru/android/2-creating-custom-android-views-part-2.html" \o "Как создать custom View для Android. Ч2.: Padding. Как нарисовать свой собственный контент)

 Pavel Bobkov

Android

11 Август 2015

[**android**](http://codeandlife.ru/component/tags/tag/2-android.html) [**tutorial**](http://codeandlife.ru/component/tags/tag/4-tutorial.html) [**view**](http://codeandlife.ru/component/tags/tag/6-view.html)

Новый View компонент

Иногда расширения стандартного View класса не достаточно чтобы решить требуемую задачу и вам придется рисовать содержимое view компонента самостоятельно. В этом руководстве мы создадим view, который рисует график, используя класс Canvas, а также немного изучим как работает view dimensions и padding.

Если вы к этому еще не готовы, прочитайте [предыдущую часть](http://codeandlife.ru/android/1-creating-custom-android-views-part-1.html).

Рисуем первые пиксели

Если вы хотите нарисовать в custom view что-то свое, лучше всего просто расширить класс View. Это самый базовый блок пользовательского интерфейса (UI) и также  полностью функциональный класс, хотя кое-чего в нем не хватает.

Итак, мы начнем с создания нового класса, который расширяет класс View.

|  |  |
| --- | --- |
| 1  2  3  4  5 | public class LineChartView extends View {     public LineChartView(Context context, AttributeSet attrs) {          super(context, attrs);      }  } |

Чтобы нарисовать первые пиксели, нам нужно просто переопределить метод onDraw(), в этом методе мы получаем объект canvas, который можно использовать для рисования. В данном случае нам не нужно вызывать onDraw() метод суперкласса, потому что для класса View этот метод просто пустой.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @Override  protected void onDraw(Canvas canvas) {      Paint paint = new Paint();      paint.setStyle(Style.STROKE);      paint.setColor(0xFF33B5E5);      paint.setStrokeWidth(4);      canvas.drawLine(0, 0, getWidth(), getHeight(), paint);  } |

Этот код рисует голубую линию толщиной 4 пикселя, которая начинается в точке (0, 0) и заканчивается в точке (getWidth(), getHeight()). Класс Paint определяет каким образом все это рисуется. Хотя объект paint  позволяет создавать достаточно много интересных эффектов, здесь он используется только для установки цвета и стиля рисования.

Обратите внимание, при рисовании верхняя левая часть view - это (0, 0), независимо от того, где view находится на экране. View класс имеет методы getLeft() и getTop(), но они возвращают позицию view относительно родительского view, поэтому их не следует использовать при рисовании.

Итак, мы создали свой собственный view, который рисует линию. Это не слишком полезно, но мы по крайней мере с чего то начали.

Добавляем padding

Когда вы создаете XML layout, вы наверняка используете в различных view такой параметр как padding. Однако, если вы попытаетесь использовать padding для view, который мы только что создали, вы увидите, что это не работает. Это связано с тем,  что мы не учли его при рисовании.

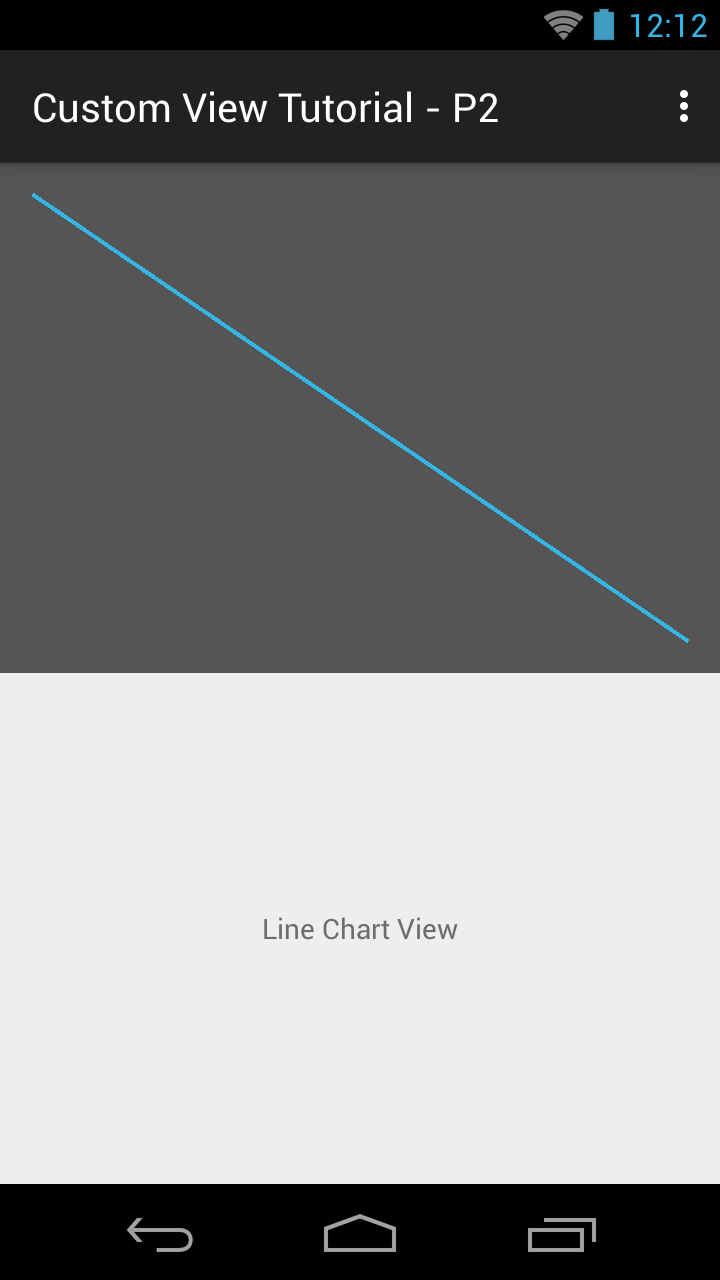
Все довольно просто, padding входит в ширину нашего view. Если view шириной 100 пикселей, а padding по 10 пикселей с обоих сторон, тогда ширина, доступная для рисования, составит 80 пикселей (то же самое, конечно, применимо и к высоте). Когда мы используем функцию getWidth(), то получаем ширину, включающую padding. Используя методы getPaddingTop(), getPaddingBottom(), getPaddingLeft() и getPaddingRight() мы можем получить величину padding.

Для того чтобы добавить поддержку padding при рисовании нашей линии, ее нужно начинать в точке (getPaddingLeft(), getPaddingTop()) и заканчивать в нижнем правом углу минус padding.

Измененный onDraw() метод будет выглядеть так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | @Override  protected void onDraw(Canvas canvas) {      paint.setStyle(Style.STROKE);      paint.setColor(0xFF33B5E5);      paint.setStrokeWidth(4);        int left = getPaddingLeft();      int top = getPaddingTop();      int right = getWidth() - getPaddingRight();      int bottom = getHeight() - getPaddingBottom();      canvas.drawLine(left, top, right, bottom, paint);  } |

Теперь изменения параметра padding в XML файле будут иметь должный эффект.



Я установил темно серый цвет фона view. Как видно из рисунка, фон отображается на весь view и даже за пределами padding. Это нормальное поведение.

Необходимость поддержки параметра - padding зависит от того, как вы планируете использовать ваш view. Тем не менее, я настоятельно рекомендую вам добавить поддержку параметра padding в самом начале, даже если вы думаете, что это вам не понадобится. Добавлять padding к более сложному view компоненту, например, который имеет множество деталей и обрабатывает нажатия, может быть довольно утомительным. Поэтому решив эту задачу в самом начале, вы облегчите себе дальнейшую работы.

Рисуем линейный график

Как я говорил в начале, наш view компонент будет рисовать линейный график. Давайте перейдем к этой задаче. В первую очередь нужно определить какие-нибудь данные для отображения.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | private float[] datapoints = new float[] {};    /\*\*Устанавливает данные для графика. Точки имеют  \* положительное значение и равноудалены друг от друга по х  \* График будет масштабируемым, поэтому будет использоваться  \* вся высота view.  \*  \* @param datapoints  \*     y - значения линейного графика  \*/  public void setChartData(float[] datapoints) {      this.datapoints = datapoints.clone();  } |

Эта функция позволит пользователю view задавать данные для отображения.  Для каждой точки на графике нам нужно два значения - X и Y. Но если мы зададим X координаты как равноудаленные отрезки, то можно использовать X как индекс массива, а Y как значения массива.

Чтобы правильно отображать данные, нам нужно их масштабировать, иначе они могут не поместиться в размер view. Для масштабирования значений будет использоваться следующий метод.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | private float getYPos(float value) {          float height = getHeight() - getPaddingTop() - getPaddingBottom();          float maxValue = getMax(datapoints);            // масштабирования под высоту view          value = (value / maxValue) \* height;            // инверсия          value = height - value;            // смещение чтобы учесть padding          value += getPaddingTop();            return value;  } |

Метод получает два значения - текущую Y координату и максимальное значение среди всех данных. Первое, что нужно сделать -  выполнить масштабирование. Обратите внимание, при вычислении высоты мы учитываем padding и сверху, и снизу.

Далее мы инвертируем полученное значение. Для чего это надо? На дисплее большему значению Y соответствуют точки, которые расположены в нижней части, но для нашего графика наивысшая Y координата должна быть вверху (прим.: у дисплея Y ось направлена вниз). После инвертирования мы добавляем смещение и возвращаем вычисленное значение.

Функция для вычисления максимального значения Y я приводить не буду. Она очевидна, нужно просто перебрать массив и найти максимум.

Для X координаты нам тоже понадобится функция масштабирования.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | private float getXPos(float value) {          float width = getWidth() - getPaddingLeft() - getPaddingRight();          float maxValue = datapoints.length - 1;            // масштабирования под размер view          value = (value / maxValue) \* width;            // смещение чтобы учесть padding          value += getPaddingLeft();            return value;   } |

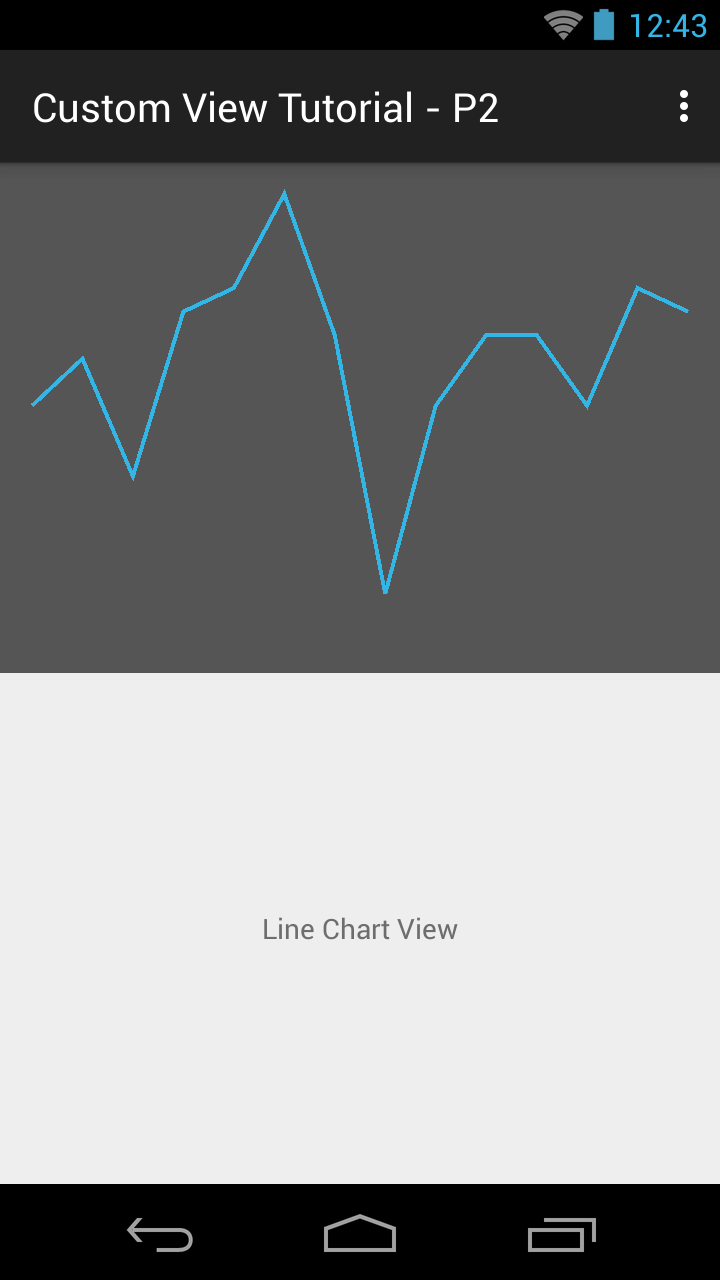
Теперь мы готовы нарисовать график. Мы можем рисовать линии между точками, но лучше решить эту задачу по-другому, используя класс Path.  Визуальной разницы никакой не будет, но это упростит нам написание кода. Метод onDraw() теперь будет выглядеть так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | protected void onDraw(Canvas canvas) {      float maxValue = getMax(datapoints);      Path path = new Path();      path.moveTo(getXPos(0), getYPos(datapoints[0]), maxValue);      for (int i = 1; i < datapoints.length; i++) {          path.lineTo(getXPos(i), getYPos(datapoints[i]), maxValue);      }        Paint paint = new Paint();      paint.setStyle(Style.STROKE);      paint.setStrokeWidth(4);      paint.setColor(0xFF33B5E5);      canvas.drawPath(path, paint);  } |

В первой части кода мы конструируем объект path, используя созданный ранее метод getYPos(). Также там используется метод getXPos(), который работает аналогичным образом, только не инвертирует значения. Создание path начинается с инициализации начальной точкой. Далее мы продлеваем path, добавляя следующие точки.

Вторая часть кода делает почти то же самое, что и раньше, только вместо метода drawLine() мы используем drawPath().

Результат (для каких-то произвольных данных) будет выглядеть так.



Это уже похоже на линейный график. Пришло время добавить еще несколько деталей.

Добавляем детали

Первая вещь, которую мы добавим - это сглаживание (anti aliasing). Это уменьшит угловатость и зазубренность линий при рисовании. Сглаживание контролируется объектом paint и его можно включить так.

|  |  |
| --- | --- |
| 1 | paint.setAntiAlias(true); |

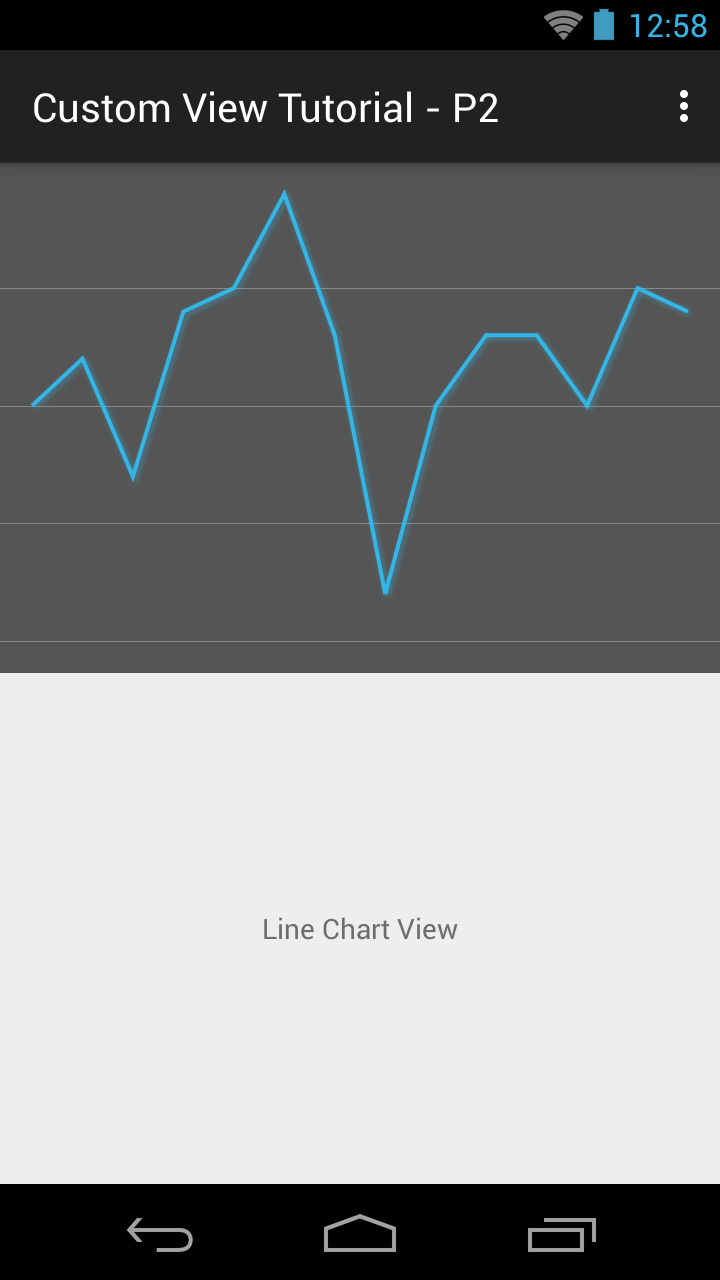
Далее мы можем улучшить внешний вид графика, добавив тени. Это делается с помощью следующего кода.

|  |  |
| --- | --- |
| 1 | paint.setShadowLayer(4, 2, 2, 0x80000000); |

Теперь все что мы нарисуем, используя paint, будет отбрасывать тени. Первый аргумент функции задает радиус размытия тени. Большее значение задает большее размытие (blur). Если установить этот аргумент 0, теневой слой будет удален. Следующие два аргумента задают смещение тени. В нашем случае тень будет смещена на 2 пикселя вправо и вниз. И последний аргумент - цвет тени и прозрачность. В данном случае установлен черный цвет и половинная прозрачность (на картинке эффект почти не заметен, но на эмуляторе его отчетливо видно).



Рисунок ниже - это результат дальнейшей доработки view компонента. Я добавил на фон горизонтальные линии, чтобы view был больше похож на график. Для рисования линий я использовал дополнительную функцию, чтобы понимать, где и в каком количестве отображать эти линии, но для данной статьи это не очень важно.



Итак, мы добавили достаточно много вещей в наш view, чтобы он выглядел подходящим образом. Используя canvas  мы можем рисовать линии, пути(paths), прямоугольники, овалы, растровые изображения и так далее. Используя объект paint мы также можем менять стиль, цвет, ширину, эффекты и другие параметры. Я рекомендую вам почитать документацию на классы[Canvas](http://developer.android.com/reference/android/graphics/Canvas.html) и [Paint](http://developer.android.com/reference/android/graphics/Paint.html" \t "_blank) и затем поиграть с различными настройками и профилями.

В следующей части мы разберемся [как реализовать анимацию в custom view](http://codeandlife.ru/android/3-creating-custom-android-views-part-3.html).

Тестовый проект для этой статьи можно скачать с [GitHub](https://github.com/CodeAndLife/Custom-view-2" \t "_blank).  
По материалам сайта [jayway](http://www.jayway.com/2012/07/03/creating-custom-android-views-part-2-how-padding-works-and-how-to-draw-your-own-content/" \t "_blank).  
Вольный перевод - Pavel Bobkov