# MPAndroidChart Developer Document

## 1、入门

本章介绍使用此库的基本设置。

**添加依赖**

首先，将此库的依赖项添加到项目中。如何执行此操作在此存储库的用法部分中进行了描述。Gradle 是使用此库作为依赖项的推荐方法。

**创建视图**

要使用 LineChart，BarChart，ScatterChart，CandleStickChart，PieChart，BubbleChart 或 RadarChart，请在.xml 中定义它：

```
<com.github.mikephil.charting.charts.LineChart
    android:id="@+id/chart"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

然后从您的 Activity，Fragment 或其他内容中检索它：

```
// in this example, a LineChart is initialized from xml
LineChart chart = (LineChart) findViewById(R.id.chart);
```

或者在代码中创建它（然后将其添加到布局中）：

```
// programmatically create a LineChart
LineChart chart = new LineChart(Context);

// get a layout defined in xml
RelativeLayout rl = (RelativeLayout) findViewById(R.id.relativeLayout);
rl.add(chart); // add the programmatically created chart
```

**添加数据**

拥有图表实例后，您可以创建数据并将其添加到图表中。此示例使用 LineChart，其中 Entry 类表示图表中具有 x 和 y 坐标的单个条目。其他图表类型（例如 BarChart）为此目的使用其他类（例如 BarEntry）。

要向图表添加数据，请将您拥有的每个数据对象包装到 Entry 对象中，如下所示：

```
YourData[] dataObjects = ...;

List<Entry> entries = new ArrayList<Entry>();

for (YourData data : dataObjects) {

    // turn your data into Entry objects
    entries.add(new Entry(data.getValueX(), data.getValueY()));
}
```

下一步，您需要将创建的 List <Entry>添加到 LineDataSet 对象。DataSet 对象保存属于一起的数据，

并允许对该数据进行单独设计。以下使用的“标签”仅具有描述性目的，并在图例中显示（如果已启用）。

```
LineDataSet dataSet = new LineDataSet(entries, "Label"); // add entries to dataset
dataSet.setColor(...);
dataSet.setValueTextColor(...); // styling, ...
```

最后一步，您需要将创建的 LineDataSet 对象（或多个对象）添加到 LineData 对象。此对象包含由 Chart 实例表示的所有数据，并允许进一步样式化。创建数据对象后，您可以将其设置为图表并刷新它：

```
LineData lineData = new LineData(dataSet);
chart.setData(lineData);
chart.invalidate(); // refresh
```

请考虑上面的场景一个非常基本的设置。有关更详细的说明，请参阅设置数据部分，其中说明了如何根据示例将数据添加到各种图表类型。

### 样式

有关图表表面和数据的设置和样式的信息，请访问常规设置和样式部分。有关各个图表类型的更具体的样式和设置，请查看特定设置和样式 Wiki 页面。

## 2、与图表的交互

该库允许您完全自定义与图表视图的可能触摸（和手势）交互，并通过回调方法对交互作出反应。

### 启用/禁用交互

- setTouchEnabled（boolean enabled）：允许启用/禁用与图表的所有可能的触摸交互。
- setDragEnabled（boolean enabled）：启用/禁用图表的拖动（平移）。
- setScaleEnabled（boolean enabled）：启用/禁用两个轴上图表的缩放。
- setScaleXEnabled（boolean enabled）：启用/禁用 x 轴缩放。
- setScaleYEnabled（boolean enabled）：启用/禁用 y 轴缩放。
- setPinchZoom（boolean enabled）：如果设置为 true，则启用手指触摸图片缩放。如果禁用，则可以单独缩放 x 轴和 y 轴。
- setDoubleTapToZoomEnabled（boolean enabled）：将此值设置为 false 以禁止通过双击来缩放图表。

### 图表 fling /减速

- setDragDecelerationEnabled（boolean enabled）：如果设置为 true，则图表会在触摸后继续滚动。默认值：true。
- setDragDecelerationFrictionCoef（float coef）：减速摩擦系数[0;1]间隔，较高的值表示速度将缓慢下降，例如，如果设置为 0，它将立即停止。1 是无效值，将自动转换为 0.9999。

### 突出显示值

如何允许通过点击手势和编程方式突出显示条目,这将在突出显示部分中进行描述。

### 手势回调

OnChartGestureListener：将允许您对图表上的手势做出反应：

```
public interface OnChartGestureListener {
```

```
    /**
     * Callbacks when a touch-gesture has started on the chart (ACTION_DOWN)
     *
     * @param me
     * @param lastPerformedGesture
     */
    void onChartGestureStart(MotionEvent me, ChartTouchListener.ChartGesture lastPerformedGesture);

    /**
     * Callbacks when a touch-gesture has ended on the chart (ACTION_UP, ACTION_CANCEL)
     *
     * @param me
     * @param lastPerformedGesture
     */
    void onChartGestureEnd(MotionEvent me, ChartTouchListener.ChartGesture lastPerformedGesture);

    /**
     * Callbacks when the chart is longpressed.
     *
     * @param me
     */
    public void onChartLongPressed(MotionEvent me);

    /**
     * Callbacks when the chart is double-tapped.
     *
     * @param me
     */
    public void onChartDoubleTapped(MotionEvent me);

    /**
     * Callbacks when the chart is single-tapped.
     *
     * @param me
     */
    public void onChartSingleTapped(MotionEvent me);

    /**
     * Callbacks then a fling gesture is made on the chart.
     *
     * @param me1
     * @param me2
     * @param velocityX
     * @param velocityY
     */
    public void onChartFling(MotionEvent me1, MotionEvent me2, float velocityX, float velocityY);

   /**
     * Callbacks when the chart is scaled / zoomed via pinch zoom gesture.
     *
     * @param me
     * @param scaleX scalefactor on the x-axis
     * @param scaleY scalefactor on the y-axis
     */
    public void onChartScale(MotionEvent me, float scaleX, float scaleY);

   /**
    * Callbacks when the chart is moved / translated via drag gesture.
    *
    * @param me
    * @param dX translation distance on the x-axis
    * @param dY translation distance on the y-axis
    */
    public void onChartTranslate(MotionEvent me, float dX, float dY);
}
```

只需让你的接收回调的类实现这个接口并将其设置为图表的监听器：

```
chart.setOnChartGestureListener(this);
```

### 3、突出

本节重点介绍通过 tap-gesture 和基于发行版 v3.0.0 以编程方式突出显示图表中条目的主题。

**启用/禁用突出显示**

- setHighlightPerDragEnabled（boolean enabled）：在图表上将此值设置为 true，以便在完全缩小时在图表表面上进行每次拖动时突出显示。默认值：true
- setHighlightPerTapEnabled（boolean enabled）：在图表上将此设置为 false，以防止通过点按手势突出显示值。仍可通过拖动或以编程方式突出显示值。默认值：true
- setMaxHighlightDistance（float distanceDp）：以 dp 为单位设置最大高亮距离。在图表上轻敲远于该距离的条目将不会触发高亮显示。默认值：500dp

除此之外，还可以为各个 DataSet 对象配置突出显示：

```
dataSet.setHighlightEnabled(true); // allow highlighting for DataSet

// set this to false to disable the drawing of highlight indicator (lines)
dataSet.setDrawHighlightIndicators(true);
dataSet.setHighlightColor(Color.BLACK); // color for highlight indicator
// and more...
```

**以编程方式突出显示**

- highlightValue（float x，int dataSetIndex，boolean callListener）：突出显示给定 DataSet 中给定 x 位置的值。提供-1 作为 dataSetIndex 以撤消所有突出显示。布尔标志确定应该调用选择侦听器还是不调用选择侦听器。
- highlightValue（Highlight high，boolean callListener）：突出显示由提供的 Highlight 对象表示的值。提供 null 以撤消所有突出显示。布尔标志确定应该调用选择侦听器还是不调用选择侦听器。
- highlightValues（突出显示[]高）：突出显示给定 Highlight []数组所表示的值。提供 null 或空数组以撤消所有突出显示。
- getHighlighted（）：返回一个 Highlight []数组，其中包含有关所有突出显示的条目，x-index 和 dataset-index 的信息。

**选择回调**

这个库提供了一些交互上的回调监听。其中一个是 OnChartValueSelectedListener 在回调时，用于通过触摸高亮一些值时的回调：

```
public interface OnChartValueSelectedListener {
    /**
    * Called when a value has been selected inside the chart.
    *
    * @param e The selected Entry.
    * @param h The corresponding highlight object that contains information
    * about the highlighted position
    */
    public void onValueSelected(Entry e, Highlight h);
    /**
    * Called when nothing has been selected or an "un-select" has been made.
    */
    public void onNothingSelected();
}
```

只是让应该接受回调函数的类实现这个接口，并把它作为图表的一个监听：

```
chart.setOnChartValueSelectedListener(this);
```

**高亮**

这个 Highlight 类代表与高亮显示的条目相关的所有数据，比如高亮 Entry 对象本身，它所属的 DataSet，它在绘图表面上的位置等等。它可以用来获取有关已经高亮的 Entry 信息，或用于为图表提供信息对于一个即将被高亮的 Entry。对于这个目的，本要闻类提供了两个构造函数：

```
 /** constructor for standard highlight */
 public Highlight(float x, int dataSetIndex) { ... }

 /** constructor for stacked BarEntry highlight */
 public Highlight(float x, int dataSetIndex, int stackIndex) { ... }
```

这些函数可以用来创建一个 Highlight 对象允许执行突出编程：

```
// highlight the entry and x-position 50 in the first (0) DataSet
Highlight highlight = new Highlight(50f, 0);

chart.highlightValue(highlight, false); // highlight this value, don't call listener
```

**自定义高亮**

所有以高亮手势形式的用户输入被缺省的 ChartHighlighter 类内部处理，可以用下面的方法定制实现来替换默认的 highligher：

- setHighlighter(ChartHighlighter highlighter)：为图表设置一个自定义高亮对象处理所有的高亮触摸事件。您的自定义 hignlighter 对象需要扩展 charthighlighter 类

**4、轴心**

此 Wiki 页面侧重于 AxisBaseclass，它是 XAxis（XAxis）和 YAxis（YAxis）的基类。在 v2.0.0 中引入。以下提到的方法可以应用于两个轴。轴类允许特定样式并包含（可以包含）以下组件/部件：

- 标签（以垂直（y 轴）或水平（x 轴）对齐绘制），包含轴描述值
- 所谓的“轴线”，直接绘制在标签旁边并与之平行
- “网格线”，每个都来自水平方向的轴标签
- LimitLines，允许提供特殊信息，如边框或约束

**控制应绘制哪些部分（轴）**

- setEnabled（boolean enabled）：设置轴启用或禁用。如果禁用，则无论其他任何设置如何，都不会绘制轴的任何部分。
- setDrawLabels（boolean enabled）：将此属性设置为 true 以启用绘制轴标签。
- setDrawAxisLine（boolean enabled）：如果应该绘制沿轴（轴线）的线，则将其设置为 true。
- setDrawGridLines（boolean enabled）：将此属性设置为 true 以启用绘制轴的网格线。

**自定义轴范围（最小/最大）**

- setAxisMaximum（float max）：为此轴设置自定义最大值。如果设置，则不会根据提供的数据自动计算该值。
- resetAxisMaximum（）：调用此方法以撤消先前设置的最大值。通过这样做，您将再次允许轴自动计算它的最大值。
- setAxisMinimum（float min）：为此轴设置自定义最小值。如果设置，则不会根据提供的数据自动计算该值。
- resetAxisMinimum（）：调用此方法以撤消先前设置的最小值。通过这样做，您将再次允许轴自动计算它的最小值。
- setStartAtZero（boolean enabled）：Deprecated- Use setAxisMinValue（...）orsetAxisMaxValue（...）。
- setInverted（boolean enabled）：如果设置为 true，则此轴将被反转，这意味着最高值将位于底部，最低值将位于顶部。
- setSpaceTop（float percent）：设置图表中最高值的顶部间距（以总轴 - 范围的百分比表示）与轴上的最高值进行比较。
- setSpaceBottom（float percent）：设置图表中最低值的底部间距（以总轴 - 范围的百分比表示）与轴上的最低值进行比较。
- setShowOnlyMinMax（boolean enabled）：如果启用，此轴将仅显示其最小值和最大值。这将忽略/覆盖定义的标签计数（如果不强制）。
- setLabelCount（int count，boolean force）：设置 y 轴的标签数。请注意，此数字不是固定的（如果 force == false），并且只能近似。如果强制启用（true），则绘制精确指定的标签计数 - 这可能导致轴上的数字不均匀。
- setPosition（YAxisLabelPosition pos）：设置应绘制轴标签的位置。INSIDE_CHART 或 OUTSIDE_CHART。
- setGranularity（float gran）：设置 y 轴值之间的最小间隔。这可以用于避免在放大到为轴设置的小数位数不再允许区分两个轴值的点时出现值重复。
- setGranularityEnabled（boolean enabled）：启用粒度特征，在放大时限制 y 轴的间隔。默认值：false

**造型/修改轴**

- setTextColor（int color）：设置轴标签的颜色。
- setTextSize（float size）：以 dp 为单位设置轴标签的文本大小。
- setTypeface（Typeface tf）：为轴标签设置自定义字体。
- setGridColor（int color）：设置此轴的网格线的颜色。
- setGridLineWidth（float width）：设置此轴的网格线的宽度。
- setAxisLineColor（int color）：设置此轴的轴线颜色。
- setAxisLineWidth（float width）：设置此轴的轴线宽度。
- enableGridDashedLine（float lineLength，float spaceLength，float phase）：使网格线以虚线

模式绘制，例如：喜欢这个 ” - - - - - -“ 。 “lineLength” 控制线条的长度， “spaceLength” 控制线条之间的空间， “phase” 控制起始点。

**格式化轴值**

要格式化轴值，可以使用 ValueFormatter 类。一个详细的格式化轴值教程可以在此(https://weeklycoding.com/mpandroidchart-documentation/formatting-data-values/)找到，这里也有一个视频教程(https://weeklycoding.com/downloads/working-with-xaxis-and-yaxis/)覆盖了格式化轴值

**限制线**

两个轴都支持所谓的 LimitLines，它允许表达特殊信息，如边界或约束。添加到 YAxis 的 LimitLines 绘制在水平方向上，在添加到 XAxis 时在垂直方向上绘制。这是您从轴添加和删除 LimitLines 的方法：

- addLimitLine（LimitLine l）：向此轴添加新的 LimitLine。
- removeLimitLine（LimitLine l）：从此轴移除指定的 LimitLine。
- 更多添加/删除方法同样也可用。
- setDrawLimitLinesBehindData（boolean enabled）：允许控制 LimitLines 和实际数据之间的 z 顺序。如果将此值设置为 true，则会在实际数据后面绘制 LimitLines，否则将在前面绘制。默认值：false

边界或限制线（类 LimitLine）是（顾名思义）简单明了的线，用于为用户提供附加信息。

举个例子，在用户登录用的应用程序中，图表可能会显示各种血压测量结果。 为了让用户知道超过 140mmHg 的血压是被认为是一个健康风险的用户，可以添加一个 140 的 LimitLine 来提供该信息。示例代码 。

示例代码

```
YAxis leftAxis = chart.getAxisLeft();

LimitLine ll = new LimitLine(140f, "Critical Blood Pressure");
ll.setLineColor(Color.RED);
ll.setLineWidth(4f);
ll.setTextColor(Color.BLACK);
ll.setTextSize(12f);
// .. and more styling options

leftAxis.addLimitLine(ll);
```

## 5、XAxis

XAxis 是 AxisBase 的子类，它继承了许多样式和便利方法。

XAxis 类（在 2.0.0 之前的版本中称为 XLabels），是与水平轴相关的所有内容的数据和信息容器。每个 Line-，Bar-，Scatter-，CandleStick-和 RadarChart 都有一个 XAxis 对象。

XAxis 类允许特定样式，并包含（可以包含）以下组件/部件：

- 所谓的 “轴线”，直接绘制在标签旁边并与之平行
- “网格线”，每个都垂直于轴标签

要获取 XAxis 类的实例，请执行以下操作：

```
XAxis xAxis = chart.getXAxis();
```

**自定义轴值**

- setLabelRotationAngle（float angle）：设置绘制 x 轴标签的角度（以度为单位）。
- setPosition（XAxisPosition pos）：设置 XAxis 应出现的位置。在 TOP，BOTTOM，BOTH_SIDED，TOP_INSIDE 或 BOTTOM_INSIDE 中选择。

**Example Code**

```
XAxis xAxis = chart.getXAxis();
xAxis.setPosition(XAxisPosition.BOTTOM);
xAxis.setTextSize(10f);
xAxis.setTextColor(Color.RED);
xAxis.setDrawAxisLine(true);
xAxis.setDrawGridLines(false);
// set a custom value formatter
xAxis.setValueFormatter(new MyCustomFormatter());
// and more...
```

## 6、YAxis

Y 轴是 AxisBase 的子类。这个 wiki 条目只描述了 YAxis，而不是它的超类。

YAxis 类（在早于 2.0.0 的版本中称为 YLabels）是与垂直轴相关的所有内容的数据和信息容器。每个 Line-，Bar-，Scatter 或 CandleStickChart 都有一个左右 YAxis 对象，分别负责左轴或右轴。RadarChart 只有一个 YAxis。默认情况下，图表的两个轴都已启用并将被绘制。

为了获取一个 YAxis 类的实例，调用下面方法的其中一个即可

```
YAxis leftAxis = chart.getAxisLeft();
YAxis rightAxis = chart.getAxisRight();

YAxis leftAxis = chart.getAxis(AxisDependency.LEFT);

YAxis yAxis = radarChart.getYAxis(); // this method radarchart only
```

在运行时，使用 public AxisDependency getAxisDependency（）来确定这个轴所代表的是图表的哪一侧。

在为图表设置数据之前，需要应用影响轴值范围的自定义。

**Axis Dependency**

默认情况下，添加到图表的所有数据都会绘制在图表的左侧 YAxis 上。如果未进一步指定和启用，则调整右 YAxis 以表示与左轴相同的比例。

如果您的图表需要支持不同的轴刻度，您可以通过设置数据应绘制的轴来实现。这可以通过更改 DataSet 对象的 AxisDependency 来完成：

```
LineDataSet dataSet = ...; // get a dataset
dataSet.setAxisDependency(AxisDependency.RIGHT);
```

设置此项将更改绘制数据的轴。

**零线**

除了在 YAxis 上沿着每个值水平起源的网格线之外，还有一个所谓的零线，它在轴上的零（0）值处绘

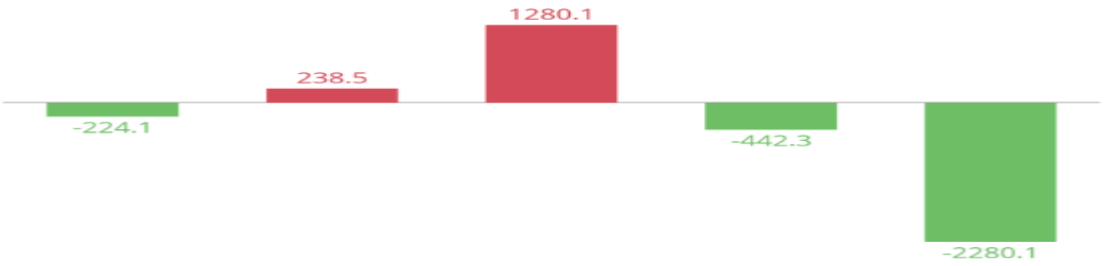
制，并且类似于网格线但可以单独配置。

- setDrawZeroLine（boolean enabled）：启用/禁用绘制零线。
- setZeroLineWidth（float width）：设置零线的行宽。
- setZeroLineColor（int color）：设置零线应具有的颜色。

零线示例代码：

```
// data has AxisDependency.LEFT
YAxis left = mChart.getAxisLeft();
left.setDrawLabels(false); // no axis labels
left.setDrawAxisLine(false); // no axis line
left.setDrawGridLines(false); // no grid lines
left.setDrawZeroLine(true); // draw a zero line
mChart.getAxisRight().setEnabled(false); // no right axis
```

上面的代码将产生如下图所示的零线。不绘制轴值，不绘制网格线或轴线，只绘制零线。




**更多示例代码**

```
YAxis yAxis = mChart.getAxisLeft();
yAxis.setTypeface(...); // set a different font
yAxis.setTextSize(12f); // set the text size
yAxis.setAxisMinimum(0f); // start at zero
yAxis.setAxisMaximum(100f); // the axis maximum is 100
yAxis.setTextColor(Color.BLACK);
yAxis.setValueFormatter(new MyValueFormatter());
yAxis.setGranularity(1f); // interval 1
yAxis.setLabelCount(6, true); // force 6 labels
//... and more
```

## 7、设置数据

如果想要给图表添加数据需要使用以下方式:。

```
public void setData(ChartData data) { ... }
```

基类 ChartData（ChartData）类封装了图表渲染期间所需的所有数据和信息。对于每种类型的图表，都存在 ChartData 的不同子类（例如 LineData），用于设置图表的数据。在构造函数中，您可以移交 List <? 将 IDataSet>扩展为要显示的值。下面是 LineData 类（扩展 ChartData）的示例，它用于向 LineChart 添加数据：

```
/** List constructor */
public LineData(List<ILineDataSet> sets) { ... }

/** Constructor with one or multiple ILineDataSet objects */
```

- `public LineData(ILineDataSet...) { ... }`

那么，什么是 DataSet 以及为什么需要它？这实际上非常简单。一个 DataSet 对象代表图表内属于一起的一组条目（例如，Entry 类）。它被设计旨在逻辑上分离图表中的不同值组。对于每种类型的图表，继承自 DataSet（例如 LineDataSet）的不同对象，其允许特定样式。

例如，您可能希望在 LineChart 中展示一年内两家不同公司的季度收入。在这种情况下，建议创建两个不同的 LineDataSet 对象，每个对象包含四个值（每个季度一个）。

当然，也可以只提供一个包含两个公司的所有 8 个值的 LineDataSet 对象。

那么如何设置 LineDataSet 对象呢?

```
public LineDataSet(List<Entry> entries, String label) { ... }
```

在查看构造函数（不同的构造函数可用）时，可以看到 LineDataSet 需要一个类型为 Entry 的 List 和一个用于描述 LineDataSet 的 String 并且用于 Legend 的标签。此外，此标签可用于在 LineData 对象中的其他 LineDataSet 对象中查找此 LineDataSet。

Entry 类型列表封装了图表的所有值。Entry 对象是图表中具有 x 值和 y 值的条目的附加包装器：

```
public Entry(float x, float y) { ... }
```

把所有的放在一起（例如，两家公司一年内的季度收入）：

首先，创建持有您的值的 Entry 类型列表：

```
List<Entry> valsComp1 = new ArrayList<Entry>();
List<Entry> valsComp2 = new ArrayList<Entry>();
```

然后，使用 Entry 对象填充列表。确保条目对象包含 x 轴的正确索引。(当然，这里可以使用循环，在这种情况下，循环的计数器变量可以是 x 轴上的索引)。

```
Entry c1e1 = new Entry(0f, 100000f); // 0 == quarter 1
valsComp1.add(c1e1);
Entry c1e2 = new Entry(1f, 140000f); // 1 == quarter 2 ...
valsComp1.add(c1e2);
// and so on ...

Entry c2e1 = new Entry(0f, 130000f); // 0 == quarter 1
valsComp2.add(c2e1);
Entry c2e2 = new Entry(1f, 115000f); // 1 == quarter 2 ...
valsComp2.add(c2e2);
//...
```

现在我们有了 Entry 对象列表，可以创建 LineDataSet 对象：

```
LineDataSet setComp1 = new LineDataSet(valsComp1, "Company 1");
setComp1.setAxisDependency(AxisDependency.LEFT);
LineDataSet setComp2 = new LineDataSet(valsComp2, "Company 2");
setComp2.setAxisDependency(AxisDependency.LEFT);
```

通过调用 setAxisDependency（...），可以指定 DataSet 应绘制的轴。最后但并非最不重要的是，我们创建了一个 IDataSets 列表并构建了我们的 ChartData 对象：

```
    // use the interface ILineDataSet
    List<ILineDataSet> dataSets = new ArrayList<ILineDataSet>();
    dataSets.add(setComp1);
    dataSets.add(setComp2);

    LineData data = new LineData(dataSets);
    mLineChart.setData(data);
    mLineChart.invalidate(); // refresh
```

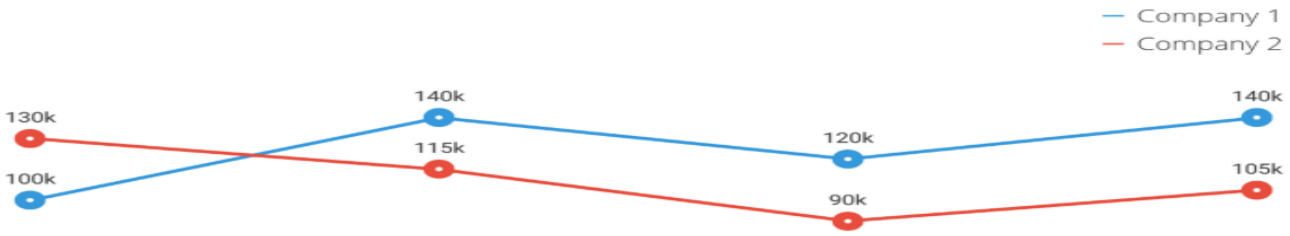
调用 invalidate（）后，将刷新图表并绘制提供的数据。

如果我们想要为 x 轴添加更多描述性值（而不是不同季度的 0 到 3 之间的数字），我们可以通过使用 ValueFormatter 类来实现。此类允许自定义 XAxis 上绘制的值的样式。在此示例中，formatter 可能如下所示：

```
// the labels that should be drawn on the XAxis
final String[] quarters = new String[] { "Q1", "Q2", "Q3", "Q4" };

IAxisValueFormatter formatter = new IAxisValueFormatter() {

    @Override
    public String getFormattedValue(float value, AxisBase axis) {
        return quarters[(int) value];
    }

    // we don't draw numbers, so no decimal digits needed
    @Override
    public int getDecimalDigits() {  return 0; }
};

XAxis xAxis = mLineChart.getXAxis();
xAxis.setGranularity(1f); // minimum axis-step (interval) is 1
xAxis.setValueFormatter(formatter);
```

关于 formatting values 的详细信息，请参见此处。

如果应用了其他样式，则此示例产生的 LineChart 外观应类似于以下示例：




设置正常 BarChart，ScatterChart，BubbleChart 和 CandleStickChart 的数据类似于 LineChart。一个特例是带有多个（分组）的条形 BarChart，下面将对此进行说明。

**条目的顺序**

请注意，该库不正式支持未按条目的 x 位置升序排列的条目列表绘制 LineChart 数据。 以未排序的方式添加条目可能会导致正确绘制，但也可能导致意外的行为。 条目对象列表可以手动排序，也可以使用 EntryXComparator 进行排序。

可以手动或使用 EntryXComparator 对 Entry 对象列表进行排序：

```
List<Entry> entries = ...;
Collections.sort(entries, new EntryXComparator());
```

之所以需要这样做是因为该库使用二进制搜索算法以获得更好的性能，仅用于排序列表。

### BarChart

为 BarChart 设置数据的方式与 LineChart 非常相似。主要区别是需要用于设置数据的数据对象（例如 BarEntry 而不是 Entry）。除此之外，BarChart 还有不同的样式选择。

请考虑以下使用数据填充 BarChart 的示例：

```
List<BarEntry> entries = new ArrayList<>();
entries.add(new BarEntry(0f, 30f));
entries.add(new BarEntry(1f, 80f));
entries.add(new BarEntry(2f, 60f));
entries.add(new BarEntry(3f, 50f));
                                    // gap of 2f
entries.add(new BarEntry(5f, 70f));
entries.add(new BarEntry(6f, 60f));

BarDataSet set = new BarDataSet(entries, "BarDataSet");
```

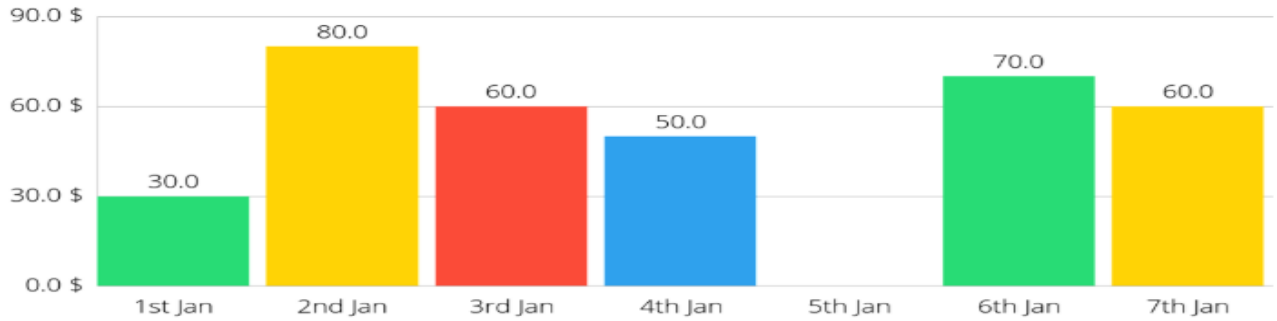
在上面的示例中，创建了五个 BarEntry 对象并将其添加到 BarDataSet。请注意，第四个和第五个条目之间的 x 位置存在“2”的间隙。在此示例中，此间隙用于解释 BarChart 中柱的定位如何工作。本教程末尾的屏幕截图将显示给定数据的结果 BarChart。

下一步，需要创建一个 BarData 对象：

```
BarData data = new BarData(set);
data.setBarWidth(0.9f); // set custom bar width
chart.setData(data);
chart.setFitBars(true); // make the x-axis fit exactly all bars
chart.invalidate(); // refresh
```

在上面的代码片段中，创建了一个 BarData 对象。当创建图表的 BarEntry 对象时，我们在每个条形（中心）之间的 x 轴上留下了“1f”的空间。通过将条宽设置为 0.9f，我们有效地在每个条之间创建 0.1f 的空间。setFitBars（true）调用将告诉图表调整它的 x 轴值范围以完全适合所有条形，并且两侧不会切断条形。

创建 BarData 对象后，我们将其设置为图表并刷新。结果应该看起来接近下面显示的结果：




### 分组 BarChart

从版本 v3.0.0 开始，MPAndroidChart 支持明确分组的绘制 bar（在这种情况下，库将处理 x 位置）或用户定义，这意味着用户可以通过更改 x 位置将条放置在他想要的任何位置。

本节将重点介绍显式分组 BarChart，这意味着库将处理条形的 x 位置。请考虑以下示例设置：

```
YourData[] group1 = ...;
YourData[] group2 = ...;

List<BarEntry> entriesGroup1 = new ArrayList<>();
List<BarEntry> entriesGroup2 = new ArrayList<>();

// fill the lists
for(int i = 0; i < group1.length; i++) {
    entriesGroup1.add(new BarEntry(i, group1.getValue()));
    entriesGroup2.add(new BarEntry(i, group2.getValue()));
}

BarDataSet set1 = new BarDataSet(entriesGroup1, "Group 1");
BarDataSet set2 = new BarDataSet(entriesGroup2, "Group 2");
```

在这个示例中，我们将有两组条形图，每条条形图由一个单独的 BarDataSet 表示。在显式（库处理）组的情况下，条目的实际 x 位置无关紧要。基于条目列表中 BarEntry 的位置执行分组。

```
float groupSpace = 0.06f;
float barSpace = 0.02f; // x2 dataset
float barWidth = 0.45f; // x2 dataset
// (0.02 + 0.45) * 2 + 0.06 = 1.00 -> interval per "group"

BarData data = new BarData(set1, set2);
data.setBarWidth(barWidth); // set the width of each bar
barChart.setData(data);
barChart.groupBars(1980f, groupSpace, barSpace); // perform the "explicit" grouping
barChart.invalidate(); // refresh
```

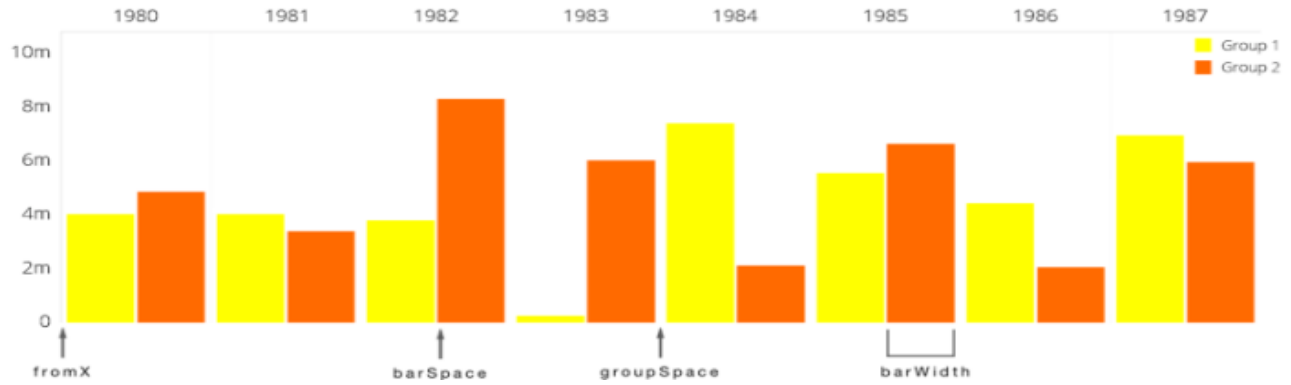
在上面的代码片段中，BarDataSet 对象被添加到 BarChart 中。groupBars（...）方法执行两个 BarDataSet 对象的分组。该方法采用以下参数：

```
public void groupBars(float fromX, float groupSpace, float barSpace) { ... }
```

fromX 参数确定分组条应该从 XAxis 开始的位置（在本例中为“1980”），groupSpace 确定每组条之间留下的空间，barSpace 确定组中各个条之间的间距。基于这些参数，groupBars（...）方法将 XAxis 上每个条形的位置改变为分组外观，从而保留各个 BarEntry 对象的顺序。

每个组在 XAxis 上具有的“间隔”（占用空间）也由 groupSpace 和 barSpace 参数以及 barWidth 定义。

结果应该看起来像这样：



当然，通过简单地手动将各个条正确地放置在 XAxis 上，也可以在不使用 groupBars（...）方法的情况下实现分组的 BarChart。

为了确保 XAxis 的标签位于组上方，如上面的屏幕截图所示，您可以使用 setCenterAxisLabels（...）方法：

```
XAxis xAxis = chart.getXAxis();
xAxis.setCenterAxisLabels(true);
```

### 堆积的 BarChart

堆叠的 BarChart 设置与普通的 BarChart 完全相似，但创建单个 BarEntry 对象的方式除外。如果是堆叠条形，则必须使用 BarEntry 的不同构造函数：

```
public BarEntry(float x, float [] yValues) { ... }
```

此构造函数允许提供多个 y 值，表示每个条的“堆栈”的值。请考虑以下示例对象：

```
BarEntry stackedEntry = new BarEntry(0f, new float[] { 10, 20, 30 });
```

该 BarEntry 由三个值组成，其“高度”为“10”，“20”和“30”。

### PieChart

与其他图表类型不同，PieChart 以 PieEntry 对象的形式获取数据。这些对象的构造函数如下所示：

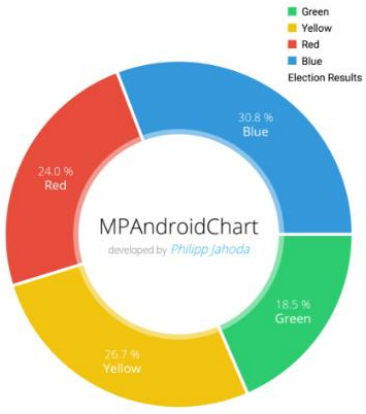
```
public PieEntry(float value, String label) { ... }
```

构造函数的第一个参数用于实际的“值”，应该在 PieChart 中将其绘制为饼图。名为“label”的第二个 String 参数用于提供切片的其他描述。请考虑以下示例 PieChart 设置：

```
List<PieEntry> entries = new ArrayList<>();

entries.add(new PieEntry(18.5f, "Green"));
entries.add(new PieEntry(26.7f, "Yellow"));
entries.add(new PieEntry(24.0f, "Red"));
entries.add(new PieEntry(30.8f, "Blue"));

PieDataSet set = new PieDataSet(entries, "Election Results");
PieData data = new PieData(set);
pieChart.setData(data);
pieChart.invalidate(); // refresh
```

PieEntry 对象不保存 x 位置的值，因为图表中显示的 PieEntry 对象的顺序由它们在条目列表中的顺序确定。

添加一些额外的样式时，生成的 PieChart 与上面使用的数据可能类似于：



## 8、Setting Colors

从 v1.4.0 版本开始，不再需要（不建议使用）用于在以前的版本中设置颜色的 ColorTemplate 对象。尽管如此，它仍然保留所有预定义的颜色数组（例如 ColorTemplate.VORDIPLOM_COLORS），并提供了方便的方法将颜色从资源（资源整数）转换为“真实”颜色。

现在可以通过 DataSet 对象直接指定颜色而不是 ColorTemplate，这允许为每个 DataSet 单独设置样式。

在这个简短的例子中，我们有两个不同的 LineDataSet 对象，代表两家公司的季度收入（之前在设置数据教程中提到过），我们现在想要设置不同的颜色。

我们想要什么：

“公司 1”的值应由“红色”的四种不同变体表示

“公司 2”的值应由“绿色”颜色的四种不同变体表示

这就是代码的样子：

```
LineDataSet setComp1 = new LineDataSet(valsComp1, "Company 1");
  // sets colors for the dataset, resolution of the resource name to a "real" color is
done internally
setComp1.setColors(new int[] { R.color.red1, R.color.red2, R.color.red3,
R.color.red4 }, Context);

LineDataSet setComp2 = new LineDataSet(valsComp2, "Company 2");
setComp2.setColors(new int[] { R.color.green1, R.color.green2, R.color.green3,
R.color.green4 }, Context);
```

除此之外，还有许多其他方法可以为 DataSet 设置颜色。这是完整的文档：

- setColors（int [] colors，Context c）：设置在此 DataSet 之前应该使用的颜色。只要 DataSet 表示的条目数高于 colors 数组的大小，就会重复使用颜色。您可以使用“new int [] {R.color.red，R.color.green，...}”为此方法提供颜色。在内部，使用 getResources（）。getColor（...）解析颜色。
- setColors（int [] colors）：设置在此 DataSet 之前应使用的颜色。只要 DataSet 表示的条目数高于 colors 数组的大小，就会重复使用颜色。在将它们添加到 DataSet 之前，请确保已准备好颜色（通过调用 getResources（）。getColor（...））。
- setColors（ArrayList <Integer> colors）：设置在此 DataSet 之前应使用的颜色。只要 DataSet 表示的条目数高于 colors 数组的大小，就会重复使用颜色。在将它们添加到 DataSet 之前，请确保已准备好颜色（通过调用 getResources（）。getColor（...））。
- setColor（int color）：设置应该用于此 DataSet 的一种颜色和唯一颜色。在内部，这将重新创建颜色数组并添加指定的颜色。

ColorTemplate 示例：

```
LineDataSet set = new LineDataSet(...);
set.setColors(ColorTemplate.VORDIPLOM_COLORS);
```

如果没有为 DataSet 设置颜色，则使用默认颜色。

## 9、ValueFormatter 接口

自 v1.6.2 起可用 - 在 v2.1.4 中更改（改进）

ValueFormatter 类可用于创建定制的格式化程序类，该类允许在绘制图表之前以特定方式（从 DataSet 或从轴）格式化图表中的值。

要使用 ValueFormatter，只需创建一个扩展它的新类，然后覆盖所需的任何功能。 ValueFormatter 提供了不同的功能，用于格式化来自各个图表类型和轴的条目。

如果没有重写任何函数，则 ValueFormatter 将返回默认格式，该格式通常是条目（或轴标签）的 y 值的 String 表示形式。

**创建 ValueFormatter（图表和 YAxis)**

以下格式化程序为 Line-和 BarChart 条目以及格式化 axis（YAxis）值提供了自定义格式：

```
public class MyValueFormatter implements IValueFormatter {

    private DecimalFormat mFormat;

    public MyValueFormatter() {
        mFormat = new DecimalFormat("###,###,##0.0"); // use one decimal
    }

    @Override
    public String getFormattedValue(float value, Entry entry, int dataSetIndex, ViewPortHandler
viewPortHandler) {
        // write your logic here
        return mFormat.format(value) + " $"; // e.g. append a dollar-sign
    }
}
```

**创建 ValueFormatter（XAxis)**

以下格式化程序旨在使用一周中的几天来格式化 XAxis 的值。 请注意，轴值已安全地转换为整数，并用作数组索引。 另外，您需要确保数组的长度与图表在 XAxis 上显示的值范围相对应。

```
class MyXAxisFormatter : ValueFormatter() {
     private val days = arrayOf("Mo", "Tu", "Wed", "Th", "Fr", "Sa", "Su")
     override fun getAxisLabel(value: Float, axis: AxisBase?): String {
          return days.getOrNull(value.toInt()) ?: value.toString()
     }
}
```

然后，将您的格式化程序设置为 ChartData 或 DataSet 对象：

```
// usage on whole data object
lineData.setValueFormatter(new MyValueFormatter());

// usage on individual dataset object
lineDataSet.setValueFormatter(new MyValueFormatter());
```

或者，在 XAxis 或 YAxis 上设置格式化程序以格式化轴值

```
// XAxis
chart.xAxis.valueFormatter = MyXAxisFormatter()

// left YAxis
chart.leftAxis.valueFormatter = MyValueFormatter()
```

**预定义的格式化器**

- LargeValueFormatter：可用于格式化大值>“1.000”。它将诸如“1.000”的值变为“1k”，“1.000.000”将变为“1m”（百万），“1.000.000.000”将变为“1b”（十亿），并且例如一万亿的值将是例如“1 吨”。
- PercentFormatter：用于在每个值后面显示“%”符号，带有 1 个十进制数字。对 PieChart 尤其有用。50 - > 50.0%
- StackedValueFormatter：专门设计用于堆叠 BarChart 的格式化程序。它允许指定是应绘制所有堆栈值还是仅绘制最高值。

## 重要提示

从 v3.1.0 版本开始，不再有轴值和图表数据值的单独格式化程序，它们现在都由 ValueFormatter 格式化。

## 10、General Chart Settings & Styling(常规图表设置和样式)

本节重点介绍适用于所有图表类型的此库的设置和样式。

**Refreshing**

- invalidate（）：在图表上调用此方法将刷新（重绘）它。为了使图表上执行的更改生效，需要这样做。
- notifyDataSetChanged（）：让图表知道它的基础数据已经改变并执行所有必要的重新计算（偏移，图例，最大值，最小值，.....）。特别是在动态添加数据时需要这样做。

**Logging(记录)**

- setLogEnabled（boolean enabled）：将此设置为 true 将激活图表 logcat 输出。启用此功能对性能有害，如果不必要，请禁用。

**通用图表样式**

以下是您可以在图表上直接使用的一些通用样式方法：

- setBackgroundColor（int color）：设置将覆盖整个图表视图的背景颜色。此外，可以通过布局文件中的.xml 设置背景颜色。
- setDescription（String desc）：设置显示在图表右下角的描述文本。
- setDescriptionColor（int color）：设置描述文本的颜色。
- setDescriptionPosition（float x，float y）：在屏幕上以像素为单位设置描述文本的自定义位置。
- setDescriptionTypeface（Typeface t）：设置用于绘制描述文本的字体。
- setDescriptionTextSize（float size）：设置描述文本的大小，以像素为单位，最小为 6f，最大为 16f。
- setNoDataText（String text）：设置图表为空时应显示的文本。
- setDrawGridBackground（boolean enabled）：如果启用，将绘制图表绘图区域后面的背景矩形。
- setGridBackgroundColor（int color）：设置应该使用网格背景绘制的颜色。
- setDrawBorders（boolean enabled）：启用/禁用绘制图表边框（图表周围的线条）。
- setBorderColor（int color）：设置图表边框线的颜色。
- setBorderWidth（float width）：以 dp 为单位设置图表边框线的宽度。

- setMaxVisibleValueCount（int count）：设置图表上最大可见绘制值标签的数量。这仅在启用 setDrawValues（）时生效。

## 11、特定图表设置和样式(Specific Chart Settings & Styling)

在入门章节和通用样式章节中，提到了适用于所有图表类型的通用图表设置和样式方法。本章重点介绍各个图表类型的特定设置。**Line-，Bar-，Scatter-，Candle-和 BubbleChart**

- setAutoScaleMinMaxEnabled（boolean enabled）：指示是否启用 y 轴自动缩放的标志。如果启用，则每当视口更改时，y 轴都会自动调整为当前 x 轴范围的最小和最大 y 值。对于显示财务数据的图表而言，这尤其有趣。默认值：false
- ssetKeepPositionOnRotation（boolean enabled）：设置方向更改后图表应保持其位置（缩放/滚动）。默认值：false

**BarChart（条图)**

- setDrawValueAboveBar（boolean enabled）：如果设置为 true，则所有值都会在其条形图上方绘制，而不是在其顶部之下。
- setDrawBarShadow（boolean enabled）：如果设置为 true，则在每个条形后面绘制一个灰色区域，指示最大值。启用他的功能会使性能降低约 40%。
- setDrawValuesForWholeStack（boolean enabled）：如果设置为 true，则堆叠条形的所有值都是单独绘制的，而不仅仅是它们之和的总和。
- setDrawHighlightArrow（boolean enabled）：将此项设置为 true 可在突出显示时在每个条形图上方绘制突出显示箭头。

**PieChart(饼形图)**

- setDrawSliceText（boolean enabled）：将此值设置为 true 可将 x 值文本绘制到饼图切片中。
- setUsePercentValues（boolean enabled）：如果启用此选项，则图表中的值将以百分比形式绘制，而不是以其原始值绘制。然后以百分比形式提供为 ValueFormatter 格式化的值。
- setCenterText（SpannableString text）：设置在 PieChart 中心绘制的文本。较长的文本将自动“包装”以避免剪切到饼图切片中。
- setCenterTextRadiusPercent（float percent）：设置中心文本边界框的矩形半径，以饼图孔默认值 1.f（100%）的百分比表示。
- setHoleRadius（float percent）：以最大半径的百分比（max =整个图表的半径）设置饼图中心的孔半径，默认为 50%
- setTransparentCircleRadius（float percent）：设置在饼图中孔旁边绘制的透明圆的半径，以最大半径的百分比表示（max =整个图表的半径），默认为 55% - >表示比缺省中心孔大 5%
- setTransparentCircleColor（int color）：设置透明圆的颜色。
- setTransparentCircleAlpha（int alpha）：设置透明圆圈应具有的透明度（0-255）。
- setMaxAngle（float maxangle）：设置用于计算饼圈的最大角度。360f 意味着它是一个完整的 PieChart，180f 会产生一个半饼图。默认值：360f

**RadarChart（雷达图)**

- setSkipWebLineCount（int count）：允许跳过来自图表中心的 Web 行。如果有很多行，特别有用。

## 12、Legend(图例)

默认情况下，所有图表类型都支持图例，并且在为图表设置数据后将自动生成并绘制图例。 图例通常由多个条目组成，每个条目均由标签，表单/形状表示。

自动生成的图例包含的条目数取决于不同颜色（跨所有 DataSet 对象）的数目以及 DataSet 标签。 图例的标签取决于为图表中使用的 DataSet 对象设置的标签。 如果没有为 DataSet 对象指定标签，则图表将自动生成它们。 如果一个数据集使用了多种颜色，则这些颜色将被分组，并且仅由（属于）一个标签来描述。

为了自定义 Legend，你可以使用 getLengend()方法从图表中获取 Legend 对象

```
Legend legend = chart.getLegend();
```

**控制是否应绘制图例**

- setEnabled（boolean enabled）：设置启用或禁用图例。如果禁用，则不会绘制图例。

**样式/修改图例**

- setTextColor（int color）：设置图例标签的颜色。
- setTextSize（float size）：以 dp 为单位设置中图例标签的文本大小。
- setTypeface（Typeface tf）：为图例标签设置自定义字体。

**避免缠绕/剪裁**

- setWordWrapEnabled（boolean enabled）：如果启用，图例的内容将不会在图表边界之外剪切，而是创建一个新行。请注意，这会降低性能，仅适用于图表下方的图例。
- setMaxSizePercent（float maxSize）：以百分比形式设置整个图表视图的最大相对大小。默认值：0.95f（95%）

**自定义图例**

- setPosition（LegendPosition pos）：设置 LegendPosition，用于定义 Legend 应出现的位置。选择 RIGHT_OF_CHART，RIGHT_OF_CHART_CENTER，RIGHT_OF_CHART_INSIDE，BELOW_CHART_LEFT，BELOW_CHART_RIGHT，BELOW_CHART_CENTER 或 PIECHART_CENTER（仅限 PieChart），.....等等。
- setForm（LegendForm shape）：设置应该使用的 LegendForm。这是在图例标签旁边绘制的形状，其中包含图例条目所代表的 DataSet 的颜色。在 SQUARE，CIRCLE 或 LINE 之间进行选择。
- setFormSize（float size）：以 dp 为单位设置中图例表单的大小。
- setXEntrySpace（float space）：设置水平轴上图例条目之间的间距。
- setYEntrySpace（float space）：设置垂直轴上图例条目之间的间距。
- setFormToTextSpace（float space）：设置图例标签(legend-label)和相应图例表单(legend-form)之间的空格。
- setWordWrapEnabled（boolean enabled）：图例文字是否应该换行？/目前仅支持 BelowChartLeft，BelowChartRight，BelowChartCenter。/你可能想在自动换行时设置 maxSizePercent，以设置文本换行的点。

**设置自定义标签和颜色**

- setCustom（int [] colors，String [] labels）：设置自定义图例的标签和颜色数组。颜色计数应与标签数量匹配。每种颜色用于在相同索引处绘制的表单。空标签将启动一个组。（-2）颜色将避免绘制

表单这将禁用自动计算数据集中的图例标签和颜色的功能。调用 resetCustom（）重新启用自动计算（然后需要 notifyDataSetChanged（）再次自动计算图例）

- resetCustom（）：调用此方法将禁用自定义图例标签（由 setCustom（...）设置）。相反，将再次自动计算标签（在调用 notifyDataSetChanged（）之后）。
- setExtra（int [] colors，String [] labels）：设置在计算图例后将附加到自动计算颜色和标签数组末尾的颜色和标签。（如果已经计算了图例，则需要调用 notifyDataSetChanged（）以使更改生效）

**Example**

```
Legend l = chart.getLegend();
    l.setFormSize(10f); // set the size of the legend forms/shapes
    l.setForm(LegendForm.CIRCLE); // set what type of form/shape should be used
    l.setPosition(LegendPosition.BELOW_CHART_LEFT);
    l.setTypeface(...);
    l.setTextSize(12f);
    l.setTextColor(Color.BLACK);
    l.setXEntrySpace(5f); // set the space between the legend entries on the x-axis
    l.setYEntrySpace(5f); // set the space between the legend entries on the y-axis

    // set custom labels and colors
    l.setCustom(ColorTemplate.VORDIPLOM_COLORS, new String[] { "Set1", "Set2", "Set3",
"Set4", "Set5" });

    // and many more...
```

## 13、描述

图表说明是通常显示在图表右下角的文本（默认情况下）。 如果需要，它允许为显示的图表数据提供其他信息

**Usage**

以下函数可用来获取对 Description 的访问

**Styling**

以下函数允许对 Description 进行样式化和修改

//启用或禁用描述

description.setEnabled（...）；

//设置描述文字

description.setText（“...”）；

//设置说明在屏幕上的位置

description.setPosition（float x，float y）；

## 14、动态和实时数据

略

## 15、Modifying the Viewport

此库具有各种用于修改视口的方法（图表上可见的内容，视图的目标）。请注意，这些方法仅适用于 LineChart，BarChart，ScatterChart 和 CandleStickChart。

下面提到的方法由 Chart 类提供。修改视口的另一种方法是通过 ViewPortHandler 直接访问它（没有图表提供的中间安全性）。这仅适用于熟悉 API 的高级用户。

*请注意，在图表上修改视口的方法都必须在设置数据后调用，*

**抑制可见的东西**

- setVisibleXRangeMaximum（float maxXRange）：设置应该一次最大可见的区域大小（x 轴上的范围）。如果这是例如设置为 10，可以一次查看 x 轴上不超过 10 个值而无需滚动。
- setVisibleXRangeMinimum（float minXRange）：设置应该一次最小可见的区域大小（x 轴上的范围）。如果这是例如如果设置为 10，则在 x 轴上缩放最多不能超过 10 个。
- setVisibleYRangeMaximum（float maxYRange，AxisDependency axis）：设置范围大小（y-axis 轴上的范围）一次最大可见。您还需要提供此约束应适用的轴。
- setViewPortOffsets（float left, float top, float right, float bottom）：设置当前 ViewPort 的自定义偏移（实际图表窗口两侧的偏移）。设置此项将阻止图表自动计算其偏移量。使用 resetViewPortOffsets（）来撤消此操作。只有在你知道你做什么的时候才使用它。
- resetViewPortOffsets（）：重置通过 setViewPortOffsets（...）方法设置的所有自定义偏移。允许图表再次自动计算所有偏移量。
- setExtraOffsets（float left, float top, float right, float bottom）：设置要附加到自动计算的偏移的额外偏移（在图表视图周围）。这不会更改自动计算的偏移量，但会为它们增加额外的空间。

**移动视图（目标位置）**

- fitScreen（）：重置所有缩放和拖动，使图表完全适合它的边界（完全缩小）。
- moveViewToX（float xValue）：将当前视口的左侧（边）移动到指定的 x 值。
- moveViewToY（float yValue，AxisDependency axis）：将视口居中于提供的 y 轴（左侧或右侧）上的指定 y 值。
- moveViewTo（float xValue，float yValue，AxisDependency axis）：这会将当前视口的左侧移动到 x 轴上的指定 x 值，并将视口居中于提供的 y 轴上的指定 y 值（与 setVisibleXRange（...）和 setVisibleYRange（...）结合使用是有意义的。
- centerViewTo（float xValue，float yValue，AxisDependency axis）：这会将当前视口的中心移动到指定的 x 值和 y 值（与 setVisibleXRange（...）和 setVisibleYRange（...）结合使用）。

**使用动画移动视图**

（自发布 v2.2.3 起）

- moveViewToAnimated（float xValue，float yValue，AxisDependency axis，long duration）：这会将当前视口的左侧移动到 x 轴上的指定 x 值，并将视口居中放置在提供的指定 y 值上 y 轴以动画方式。
- centerViewToAnimated（float xValue，float yValue，AxisDependency axis，long duration）：这将以动画方式将当前视口的中心移动到指定的 x 值和 y 值（根据轴）。
- *注意：所有 moveViewTo（...）方法将自动使图表重绘（刷新）。无需进一步调用 invalidate（）。*

**缩放（以编程方式）**

- zoomIn（）：放大 1.4f，进入图表中心。
- zoomOut（）：从图表中心缩小 0.7f。
- zoom（float scaleX，float scaleY，float x，float y）：按给定的比例因子放大或缩小。x 和 y 是变焦中心的坐标（以像素为单位）。请记住，1f 的比例=无缩放。

- zoom（float scaleX，float scaleY，float xValue，float yValue，AxisDependency axis）：按给定的比例因子放大或缩小。xValue 和 yValue 是缩放中心的实际数据值（不是像素）。请记住，1f 的比例=无缩放。

### Zooming with animations

（自发布 v2.2.3 起）

- zoomAndCenterAnimated（float scaleX，float scaleY，float xValue，float yValue，AxisDependency axis，long duration）：缩放指定的缩放系数，并以动画方式将视口居中指定轴上的指定值。

### Full example

```
chart.setData(...); // first set data

// now modify viewport
chart.setVisibleXRangeMaximum(20); // allow 20 values to be displayed at once on the x-axis, not more
chart.moveViewToX(10); // set the left edge of the chart to x-index 10
// moveViewToX(...) also calls invalidate()
```

## 16、动画

所有图表类型都支持动画，可用于以令人敬畏的方式创建/构建图表。存在三种不同的动画方法，分别为两者或 x 轴和 y 轴设置动画：

animateX（int durationMillis）：在水平轴上设置图表值的动画，这意味着图表将在从左到右的指定时间内建立。

animateY（int durationMillis）：在垂直轴上设置图表值的动画，这意味着图表将在从下到上的指定时间内构建。

animateXY（int xDuration，int yDuration）：为水平和垂直轴设置动画，从而产生左/右 底/顶部构建。

```
mChart.animateX(3000); // animate horizontal 3000 milliseconds
// or:
mChart.animateY(3000); // animate vertical 3000 milliseconds
// or:
mChart.animateXY(3000, 3000); // animate horizontal and vertical 3000 milliseconds
```

如果调用 animate（...）（任何类型），则不需要再次调用 invalidate（）来刷新图表。

### Animation easing（动画缓和）

该库允许您将漂亮的缓动函数应用于动画。您可以在以下静态预定义 Easing.EasingOption 之间进行选择：

```
public enum EasingOption {
    Linear,
    EaseInQuad,
    EaseOutQuad,
    EaseInOutQuad,
    EaseInCubic,
    EaseOutCubic,
    EaseInOutCubic,
    EaseInQuart,
    EaseOutQuart,
    EaseInOutQuart,
    EaseInSine,
    EaseOutSine,
    EaseInOutSine,
    EaseInExpo,
    EaseOutExpo,
    EaseInOutExpo,
    EaseInCirc,
    EaseOutCirc,
    EaseInOutCirc,
    EaseInElastic,
    EaseOutElastic,
    EaseInOutElastic,
    EaseInBack,
    EaseOutBack,
    EaseInOutBack,
    EaseInBounce,
    EaseOutBounce,
    EaseInOutBounce,
}
```

基本上，有两种方法可以简化动画：

1.调用任何动画方法带有预定义的缓动选项

```
public void animateY(int durationmillis, Easing.EasingOption option);
```

使用预定义的缓动选项调用任何动画方法：

```
// animate both axes with easing
mChart.animateY(3000, Easing.EasingOption.EaseOutBack);
```

当您希望代码在 Android 3.0（API 级别 11）以下运行时，请始终使用 Easing.EasingOption 进行预定义的动画缓动。

2.自定义缓动功能:(自定义缓动功能将在 Android 3.0 下崩溃)

```
public void animateY(int durationmillis, EasingFunction function);
```

通过创建自己的缓动函数类并实现 EasingFunction 接口来创建自己的缓动函数：

```
/**
 * Interface for creating custom made easing functions.
 */
public interface EasingFunction {
    /**
     * Called everytime the animation is updated.
     * @param input - the time passed since the animation started (value between 0 and 1)
     */
     public float getInterpolation(float input);
}
```

然后以这种方式调用它（请注意，这不会在 Android 3.0 下运行并崩溃）：

```
// animate both axes with easing
mChart.animateY(3000, new MyEasingFunction());
```

## 17. MarkerView（弹出视图）IMarker Interface

从版本 v3.0.0 开始，图表中的标记（弹出视图）由 IMarker 接口表示。

### IMarker Interface

此接口允许您创建显示在图表中突出显示的条目的自定义标记视图。界面提供的方法如下：

```
public interface IMarker {

    /**
     * @return The desired (general) offset you wish the IMarker to have on the x- and
y-axis.
     *         By returning x: -(width / 2) you will center the IMarker horizontally.
     *         By returning y: -(height / 2) you will center the IMarker vertically.
     */
    MPPointF getOffset();

    /**
     * @return The offset for drawing at the specific `point`. This allows conditional
adjusting of the Marker position.
     *         If you have no adjustments to make, return getOffset().
     *
     * @param posX This is the X position at which the marker wants to be drawn.
     *             You can adjust the offset conditionally based on this argument.
     * @param posY This is the X position at which the marker wants to be drawn.
     *             You can adjust the offset conditionally based on this argument.
     */
    MPPointF getOffsetForDrawingAtPos(float posX, float posY);

    /**
     * This method enables a specified custom IMarker to update it's content every time
the IMarker is redrawn.
     *
     * @param e         The Entry the IMarker belongs to. This can also be any subclass
of Entry, like BarEntry or
     *                  CandleEntry, simply cast it at runtime.
     * @param highlight The highlight object contains information about the highlighted
value such as it's dataset-index, the
     *                  selected range or stack-index (only stacked bar entries).
     */
    void refreshContent(Entry e, Highlight highlight);

    /**
     * Draws the IMarker on the given position on the screen with the given Canvas
object.
     *
     * @param canvas
     * @param posX
     * @param posY
     */
    void draw(Canvas canvas, float posX, float posY);
}
```

### Creating a MarkerView

要创建自定义标记视图，您需要创建一个实现 IMarker 接口的新类：

```
public class YourMarkerView implements IMarker { ... }
```

从接口提供的方法中返回的内容取决于您的个人需求。 请查看上面显示的方法文档，以更好地理解。

除了实现 IMarker 接口之外，您还可以创建自己的类，并通过下面提到的预定义标记之一对其进行扩展。 这种方法比较容易，不需要实现 IMarker 接口提供的所有方法。 只能重写和自定义特定的方法。 然后，最重要的事情是重写 refreshContent（...）方法以调整标记绘制的数据。 一个简单的示例如下所示：

```
public class YourMarkerView extends MarkerView {

    private TextView tvContent;

    public MyMarkerView(Context context, int layoutResource) {
        super(context, layoutResource);

        // find your layout components
        tvContent = (TextView) findViewById(R.id.tvContent);
    }

    // callbacks everytime the MarkerView is redrawn, can be used to update the
    // content (user-interface)
    @Override
    public void refreshContent(Entry e, Highlight highlight) {

        tvContent.setText("" + e.getY());

        // this will perform necessary layouting
        super.refreshContent(e, highlight);
    }

    private MPPointF mOffset;

    @Override
    public MPPointF getOffset() {

        if(mOffset == null) {
           // center the marker horizontally and vertically
           mOffset = new MPPointF(-(getWidth() / 2), -getHeight());
        }

        return mOffset;
    }
}
```

### 获取/设置 Marker

要将标记设置为图表，请使用 setMarker（...）方法：

```
IMarker marker = new YourMarkerView();
chart.setMarker(marker);
```

要访问图表的现有标记集，请使用 getMarker（）方法：

```
IMarker marker = chart.getMarker();
```

### 预定义标记

除了创建自己的自定义标记视图外，此库还提供了一些预定义标记，以便更轻松，更快速地使这些标记包

括：

- MarkerView：基本标记。允许提供在表示标记的图表表面上呈现的布局资源。扩展此类并覆盖 refreshContent（...）方法以调整标记数据。
- MarkerImage：绘制图像的标记。允许提供在表示标记的图表表面上呈现的可绘制资源。扩展此类并覆盖 refreshContent（...）方法以调整标记数据。

### Legacy MarkerView

在 v3.0.0 之前的版本中，MarkerView 类负责在图表中突出显示的位置绘制标记。有关此课程的详细信息，请访问旧的 MarkerView 维基页面。

## 18、ChartData 类

此 Wiki 条目旨在更好地了解 MPAndroidChart 背后的数据模型。

ChartData 类是所有数据类（子类）的基类，如 LineData，BarData，...等等。它用于通过图表的 setData（...）方法为 Chart 提供数据。

```
public class LineData extends ChartData { ...
```

以下提到的方法在 ChartData 类中实现，因此可以用于所有子类。

### 样式数据

- setValueTextColor（int color）：设置此数据对象包含的所有 DataSet 的 value-text（绘制值标签的颜色）的颜色。
- setValueTextColors（List colors）：设置要用作值颜色的颜色列表。
- setValueTextSize（float size）：设置此数据对象包含的所有 DataSet 的 value-text 的大小（以 dp 为单位）。
- setValueTypeface（Typeface tf）：为此数据对象包含的所有 DataSet 的所有值标签设置字体。
- setValueFormatter（ValueFormatter f）：为此数据对象包含的所有 DataSet 设置自定义 ValueFormatter，在此处更多地在 ValueFormatter 上。
- setDrawValues（boolean enabled）：启用/禁用此数据对象包含的所有 DataSet 的绘图值（值 - 文本）。

### Getters / Convenience

- getDataSetByIndex（int index）：返回数据对象 DataSet 列表中给定索引处的 DataSet 对象。
- contains（条目条目）：检查此数据对象是否包含指定的条目。如果是，则返回 true，否则返回 false。注意：在这一点上性能非常糟糕，在性能危急情况下不要过度使用。
- contains（T dataSet）：如果此数据对象包含提供的 DataSet，则返回 true，否则返回 false。

### Clearing

- clearValues（）：清除所有 DataSet 对象的数据对象，从而清除所有条目。不删除提供的 x 值。

### Highlighting

- setHighlightEnabled（boolean enabled）：将此项设置为 true 以允许通过触摸突出显示此 ChartData 对象和所有基础 DataSet。

- setDrawVerticalHighlightIndicator（boolean enabled）：启用/禁用垂直高亮显示指示符行。如果禁用，则不绘制指标。
- setDrawHorizontalHighlightIndicator（boolean enabled）：启用/禁用水平高亮显示指示符行。如果禁用，则不绘制指标。

**Dynamic Data**

- notifyDataChanged（）：让数据对象知道它的基础数据已经改变并执行所有必要的重新计算。

有关在现有数据对象中添加和删除数据的方法，请访问动态和实时数据部分。

## 19、Chart Specific Data （ChartData 子类)

此 wiki 条目侧重于 ChartData 类的子类。此处未提及的 ChartData 的所有其他子类不提供任何特定的增强功能。

BarData **(class BarData)**

- setGroupSpace（float percent）：设置不同数据集的条形组之间的间距，以百分之一的总宽度为单位。100 =空格正好是一个条宽，默认值：80
- isGrouped（）：如果此数据对象已分组（由多个 DataSet 组成），则返回 true，否则返回 false。

**ScatterData** (class ScatterData)

- getGreatestShapeSize（）：返回此数据对象包含的所有 ScatterDataSets 中的最大 shape-size。

**PieData (class PieData)**

- getDataSet（）：返回为此数据对象设置的 PieDataSet 对象。PieData 对象不能包含多个 PieDataSet。
- setDataSet（PieDataSet set）：设置此数据对象应表示的 PieDataSet。

**BubbleData (class BubbleData)**

- setHighlightCircleWidth（float width）：设置此数据对象包含的所有 BubbleDataSet 对象在突出显示状态时围绕气泡的圆的宽度，以 dp 为单位。

**CombinedData (class CombinedData)**

此数据对象旨在包含所有其他数据对象的实例。使用 setData（...）方法为此对象提供数据。这仅用于 CombinedChart。

这就是它内部的样子：

```
public class CombinedData extends ChartData {

    // ...

    public CombinedData(List<String> xVals) { ... }

    public CombinedData(String[] xVals) { ... }

    public void setData(LineData data) { ... }

    public void setData(BarData data) { ... }

    public void setData(ScatterData data) { ... }

    public void setData(CandleData data) { ... }

    // ...
}
```

## 20、The DataSet class

DataSet 类是所有数据集类（子类）的基类，如 LineDataSet，BarDataSet，...等等。

```
public class LineDataSet extends DataSet { ...
```

DataSet 类表示图表中属于一起的一个组或条目类型（条目）。它旨在逻辑上分隔图表中不同的值组（例如，LineChart 中特定行的值，或 BarChart 中特定条形组的值）。

以下提到的方法在 DataSet 类中实现，因此可以用于所有子类。

### Styling data

- setValueTextColor（int color）：设置此 DataSet 对象的 value-text（绘制值标签的颜色）的颜色。
- setValueTextColors（List colors）：设置要用作值颜色的颜色列表。
- setValueTextSize（float size）：设置此 DataSet 对象的 value-text 的大小（以 dp 为单位）。
- setValueTypeface（Typeface tf）：为此 DataSet 对象的所有值标签设置字体。
- setValueFormatter（ValueFormatter f）：为此 DataSet 对象设置自定义 ValueFormatter，在此处为 ValueFormatter 设置更多。
- setDrawValues（boolean enabled）：启用/禁用此 DataSet 对象的绘图值（值 - 文本）。

如果整个数据对象（不是数据集）中的所有值都应该例如具有相同的颜色，您可以简单地调用 ChartData 对象上面提到的一个。

### Highlighting

- setHighlightEnabled（boolean enabled）：将此属性设置为 true 以允许通过触摸突出显示此特定 DataSet。
- setDrawVerticalHighlightIndicator（boolean enabled）：启用/禁用垂直高亮显示指示符行。如果禁用，则不绘制指标。
- setDrawHorizontalHighlightIndicator（boolean enabled）：启用/禁用水平高亮显示指示符行。如果禁用，则不绘制指标。

## 21、DataSet classes in detail

此 wiki 条目侧重于 DataSet 类的子类。此处未提及的 DataSet 的所有其他子类不提供任何特定的增强功能。

Line-，Bar-，Scatter-，Bubble-和 CandleDataSet（下面提到的方法可以用于任何提到的 DataSet 类）

- setHighLightColor（int color）：设置用于绘制高光指示符的颜色。不要忘记使用 getResources（）。getColor（...）或 Color.rgb（...）（或简称 Color.BLACK）来解析颜色。

### Line-，Bar-，Scatter-，Candle-和 RadarDataSet

- setDrawHighlightIndicators（boolean enabled）：启用/禁用垂直和水平高亮显示指示符行。为单个配置调用 setDrawVerticalHighlightIndicator（...）和 setDrawHorizontalHighlightIndicator（...）。
- setHighlightLineWidth（float width）：设置 dp 中高光线（十字准线）的宽度。

### Line- & RadarDataSet (methods only for LineDataSet and RadarDataSet)

- setFillColor（int color）：设置用于填充线条曲面的颜色。
- setFillAlpha（int alpha）：设置用于填充线表面的 alpha 值（透明度）（0-255），默认值：85,255 =完全不透明，0 =完全透明
- setFillDrawable（Drawable d）：设置应覆盖填充区域的 Drawable。这也允许使用渐变。
- setDrawFilled（boolean filled）：如果应该绘制填充的数据集（表面，区域），而不仅仅是一行，则设置为 true，禁用此选项将提高性能。请注意，此方法使用 canvas.clipPath（...）方法绘制填充区域。对于 API 级别<18（Android 4.3）的设备，应关闭图表的硬件加速 - 请参见此处。默认值：
- falsesetLineWidth（float width）：设置此 DataSet 的行宽（min = 0.2f，max = 10f）;默认 1f 注意：更细的线==更好的性能，更粗的线==更差的性能

下面提到的方法仅适用于特别提到的 DataSet 子类。

### LineDataSet (class LineDataSet)

- setCircleRadius（float size）：设置圆形值指标的大小（半径），默认大小= 4f
- setDrawCircles（boolean enabled）：将此属性设置为 true 以启用此 LineDataSet 的圆形指示器绘制，默认为 true
- setDrawCubic（boolean enabled）：如果设置为 true，则线条图线以立方体样式而不是线性线条绘制。这对性能有负面影响！默认值：false
- setCubicIntensity（float intensity）：设置立方线的强度（如果启用）。Max = 1f =非常立方，Min = 0.05f =低立方效应，默认值：0.2f
- setCircleColor（int color）：设置此数据集应具有的所有圆圈指示符的颜色。
- setCircleColors（List colors）：设置此 LineDataSet 的外圆应具有的颜色。还有各种其他方法来设置圆形颜色。
- setCircleColorHole（int color）：设置直线圆（孔）的内圆的颜色。
- setDrawCircleHole（boolean enabled）：将此值设置为 true 以允许在此数据集的每个圆中绘制一个孔。如果设置为 false，则将绘制圆圈（无孔）。
- enableDashedLine（float lineLength，float spaceLength，float phase）：允许以虚线模式绘制

线条，例如喜欢这个“- - - - - -”。“lineLength”是线条的长度，“spaceLength”是各块之间的空间长度，“phase”是偏移量，以度为单位（通常使用 0）

**BarDataSet (class BarDataSet)**

- setBarSpacePercent（float percent）：以总条宽的百分比设置条之间的间距。
- setBarShadowColor（int color）：设置用于绘制条形阴影的颜色。条形阴影是条形图后面的一个表示最大值的表面。不要使用 getResources（）。getColor（...）来设置它。或者 Color.rgb（...）。
- setHighLightAlpha（int alpha）：设置用于绘制高亮显示指示条的 alpha 值（透明度）。min = 0（完全透明），max = 255（完全不透明）。
- setStackLabels（String [] labels）：为条形图堆栈的不同值设置标签（如果有的话）。

**ScatterDataSet (class ScatterDataSet)**

- setScatterShapeSize（float size）：设置绘制的 scattershape 将具有的密度像素大小。这仅适用于非自定义形状。
- setScatterShape（ScatterShape shape）：设置在值所在的位置绘制的形状。

**CandleDataSet (class CandleDataSet)**

- setBodySpace（float space）：设置每个烛体左侧和右侧遗漏的空间，默认为 0.1f（10%），最大 0.45f，min 0f
- setShadowWidth（float width）：以 dp 为单位设置蜡烛阴影线的宽度。默认 3f。
- setShadowColor（int color）：设置蜡烛阴影线的颜色。
- setDecreasingColor（int color）：设置打开>关闭时应该用于此 DataSet 的唯一颜色。
- setIncreasingColor（int color）：设置打开<= close 时应该用于此 DataSet 的唯一颜色。
- setDecreasingPaintStyle（Paint.Style style）：在打开>关闭（填充或描边）时设置绘画样式。
- setIncreasingPaintStyle（Paint.Style style）：设置打开时的绘画样式<=关闭（填充或描边）。

**BubbleDataSet (class BubbleDataSet)**

- setHighlightCircleWidth（float width）：设置处于高亮状态时围绕气泡的圆的宽度，单位为 dp。

有关 CandleDataSet 颜色的信息：通常的 setColors（...），setColor（...），...方法仍然可以用于一次着色图表。如果需要特定颜色（用于身体，阴影.....），请使用上述方法。

**PieDataSet (class PieDataSet)**

- setSliceSpace（float degrees）：设置 dp 中饼图切片之间遗漏的空间，默认值：0 ->无空格，最大 20，最小 0（无空格）
- setSelectionShift（float shift）：设置此 DataSet 的突出显示的饼图切片远离图表中心“移位”的距离，默认为 12f

## 22、The ViewPortHandler

ViewPortHandler 类负责处理图表的视口。这意味着它负责图表视图中可见的内容，它是转换和缩放/比例级别的当前状态，图表的大小以及它的绘图区域和当前偏移。ViewPortHandler 允许直接访问所有上述属性并进行修改。

与通过 Chart 类修改视图端口不同，如此处所述，直接修改 ViewPortHandler 并不总是一种完全安全的

方式来更改可见内容并且应该由熟悉 API 的人员小心执行。使用不当可能会导致意外行为。但是，ViewPortHandler 提供了更高级的视口修改方法。

### 获得一个实例

ViewPortHandler 的实例只能通过以下方式获取：

```
ViewPortHandler handler = chart.getViewPortHandler();
```

### Scale & Translation

- getScaleX（）：返回 x 轴上的当前缩放/缩放级别。
- getScaleY（）：返回 y 轴上的当前缩放/缩放级别。
- getTransX（）：返回 x 轴上的当前平移（移动）。
- getTransY（）：返回 y 轴上的当前平移（移动）。

### 图表尺寸和内容

- getChartWidth（）：返回图表的宽度。
- getChartHeight（）：返回图表的高度。
- getContentRect（）：以 RectF 对象的形式返回当前内容区域。

可以在 JavaDocs 中找到更多方法，也可以通过研究 API 来找到。

## 23、FillFormatter

FillFormatter 接口允许自定义 LineDataSet 的填充行应该结束的位置。所有需要做的就是创建一个新类并实现 FillFormatter 接口。使用

```
public float getFillLinePosition(LineDataSet dataSet, LineDataProvider provider)
```

用于实现自定义逻辑的接口的方法，该逻辑计算单个 LineDataSet 的填充线的终点。

创建一个实现接口的类：

```
public class MyCustomFillFormatter implements FillFormatter {

    @Override
    public float getFillLinePosition(LineDataSet dataSet, LineDataProvider dataProvider) {

        float myDesiredFillPosition = ...;
        // put your logic here...

        return myDesiredFillPosition;
    }
}
```

然后将自定义格式化程序设置为 LineDataSet：

```
lineDataSet.setFillFormatter(new MyCustomFillFormatter());
```

这是 DefaultFillFormatter 的默认实现（逻辑）。

## 24、Proguard

如果您使用的是 Proguard，则需要将 MPAndroidChart 列入白名单，这需要在 Proguard 配置文件中添加以下行：

```
-keep class com.github.mikephil.charting.** { *; }
```

如果您不这样做，动画可能无法正常工作。

如果您遇到 Realm.io 类问题，请将以下内容添加到配置文件中：

```
-dontwarn io.realm.**
```

有关 Proguard 的其他信息。Additional information on Proguard.

## 25、Realm.io 数据库集成

Please refer to the official **MPAndroidChart-Realm** repository to plot data directly from Realm database with MPAndroidChart.

A detailed guide can be found here.

## 26、Custom DataSets

从 v2.2.0 开始，MPAndroidChart 允许您轻松创建自己的自定义 DataSet 对象并在图表中使用它们。

你需要做什么

- 创建自己的自定义类（例如 CustomDataSet）
- 让它扩展 BaseDataSet <？扩展 Entry>
- 让它实现您选择的 IDataSet 接口（例如 IBarDataSet） - 具体取决于您要创建的图表类型
- 实现所有（由您）所需的方法，让它们返回您选择的值

### Example

创建要在 BarChart 中使用的自定义 BarDataSet。

```
public class CustomBarDataSet extends BaseDataSet<BarEntry> implements IBarDataSet {
    // implement all by the extended class and interface required methods
}
```

在创建 CustomBarDataSet 并实现接口所需的所有方法之后，它可以在任何 BarChart 中使用，就像普通的 BarDataSet 一样。

## 27、Miscellaneous（其他）

### Chart content

- clear（）：清除所有数据的图表（通过将数据对象设置为 null）。调用 invalidate（）来刷新图表。
- clearValues（）：清除所有 DataSet 对象的图表，从而清除所有条目。不从图表中删除提供的 x 值。调用 invalidate（）来刷新图表。

- isEmpty（）：如果图表数据对象为空，或者它不包含任何条目，则返回 true。

**有用的 getter 方法**

- getData（）：将返回您为图表设置的数据对象。
- getViewPortHandler（）：将返回图表的 ViewPortHandler 对象，其中包含有关图表大小和边界（偏移，内容区域）的信息，以及有关图表当前比例（缩放）和平移（滚动）状态的信息。
- getRenderer（）：返回负责绘制图表数据的主 DataRenderer。
- getCenter（）：返回整个图表视图的中心点。
- getCenterOffsets（）：返回图表绘图区域的中心点。
- getPercentOfTotal（float value）：返回提供的值构成图表内总和值的百分比。
- getYMin（）：返回图表所包含的最低值。
- getYMax（）：返回图表所包含的最大值。
- getLowestVisibleXIndex（）：返回在图表上仍然可见的最低 x-index（x 轴上的值）。
- getHighestVisibleXIndex（）：返回在图表上仍然可见的最高 x-index（x 轴上的值）。

**一些更多的方法（Chart 类）**

- saveToGallery（String title）：将当前图表状态作为图像保存到库中。不要忘记为清单添加“WRITE_EXTERNAL_STORAGE”权限。
- saveToPath（String title，String pathOnSD）：将当前图表状态保存为指定路径的图像。不要忘记为清单添加“WRITE_EXTERNAL_STORAGE”权限。
- getChartBitmap（）：返回表示图表的 Bitmap 对象，此 Bitmap 始终包含图表的最新绘图状态。
- setHardwareAccelerationEnabled（boolean enabled）：允许启用/禁用图表的硬件加速，仅限 API 级别 11+。