常用的接口：

ChartInterface

获取最大最小值，宽高，数据、中心点，最大可见值等。

BarLineScatterCandleBubbleDataProvider （继承于ChartInterface），多了getTransformer()、isInverted()、getLowestVisibleX()、getHighestVisibleX()、getData()

BarLineScatterCandleBubbleDataProvider（继承于 ChartInterface），多出来的方法有 getTransformer()、isInverted() 、getLowestVisibleX()、getHighestVisibleX()、重写了getData()

数据： ChartData（抽象方法）,里面可以有多个 IdataSet，即是说可以有多条数据，有一个重要的方法calcMinMax(),遍历获取最大最小值，这个是获得所有子数据中的最大最小值（注意与下方的IDataSet）。notifyDataSetChange()方法调用后，都会调用calcMinMax()方法。可持有多个 IdataSet的引用。

BarLineScatterCandleBubbleData LineData

IdataSet : 获取最大最小值，计算最大最小值，根据某个条件获取index，设置左边还是右边为Y轴，设置高亮，设置ValueFormatter，设置颜色，标签。

IdataSet的实现类：BaseDataSet（抽象类），dataSet标签。

转换类 Transformer

负责转换数据为像素长度

渲染类（画图的类，传递canvas对象给它画图）。

基类 Renderer（抽象类），持有 ViewPortHandler 的引用。

DataRenderer（抽象类继承于 Renderer） 画数据、高亮，和数据的值。

AxisRenderer（抽象类继承与 Renderer），有抽象方法画坐标轴的值、网格线、坐标轴线、限制线（LimitLine）。具体方法为computeAxis()，算出最大值p1.y和最小值p2.y，computeAxisValues(float min,float max)，这个方法具体描述一下：先获取Label的数量（labelCount），再获取max – min 的绝对值（得到的是范围 range）

如果 labelCount 为0或者range小于0，或者range为无限大，则return ;

计算相邻坐标点的间隔 rawInterval = range / labelCount，再把这个值换算成小数位为0，或者是个位数位0的数（interval）。再次处理interval，如果 mAxis.isGranularityEnable() 为true(GranularityEnable 译为粒度、间隔)，interval = interval < mAxis.getGranularity() ? mAxis.getGranularity() : interval; 使 interval 保持在最小为mAxis.getGranularity() 的大小。

如果mAxis.isForceLabelsEnabled() 为true（强制Label的数量），interval = range/(labelCount - 1)，将mAxis.mEntryCount = labelCount。以min 为起点，加上 labelCount 个 interval，得到各个坐标轴的值。

如果mAxis.isForceLabelsEnable() 为false，先求出第一个数值first = interval == 0.0?0.0:Math.ceil(yMin / interval)\*interval;再求出最后一个数值，

double last = interval == 0.0 ? 0.0 : Utils.nextUp(Math.floor(yMax / interval) \* interval);再根据interval 来计算first到last之间的个数。

最基础的类是 Chart，继承于 ViewGroup，实现了 ChartInterface 接口

onDraw() 方法中，如果没有 数据，则在中间显示 “No data..”

如果有数据，调用 calculateOffsets() 抽象方法。

BarLineChartBase（抽象类） 继承于 Chart，实现了BarLineScatterCandleBubbleDataProvider（继承于ChartInterface），

onDraw() 方法中，

calculateOffsets()的实现是，计算偏移量。

首先执行 drawGridBackground() ，画出表格的背景，如果有边框则画边框。

然后执行 mAxisRendererLeft.computeAxis()，computeAxis()会执行 computeAxisValues() 方法<重要方法>，这个方法可以测量出 间距，还有数据的范围。

接着执行 mAxisRenderer的renderAxisLine，来画坐标轴线。

再接着执行 autoScale() ，在 mAutoScaleMinMaxEnabled 标志位是 true 的情况下。

再执行 mAxisRenderer.renderGridLines() 来画表格线。

再然后画 LimitLine，通过mAxisRenderer.renderLimitLine() 来画限制线。

clipRect()来固定画图的范围。

接着执行 mRenderer.drawData() 来画数据。 DataRenderer 是一个抽象类。

PieRadarChartBase 扇形图和雷达图的基类。

PieChart（继承于PieRadarChartBase），属性中的标志位mDrawEntryLabels，mDrawHole，mDrawSlicesUnderHole，mUsePercentValues，mDrawRoundedSlices，mDrawCenterText，持有的引用 PieChartRenderer（this,mAnimator,mViewPortHandler）, PieHighlighter（this）,

if (mData == null)

return;

mRenderer.drawData(canvas);

if (valuesToHighlight())

mRenderer.drawHighlighted(canvas, mIndicesToHighlight);

mRenderer.drawExtras(canvas);

mRenderer.drawValues(canvas);

mLegendRenderer.renderLegend(canvas);

drawDescription(canvas);

drawMarkers(canvas);

组合图表

CombinedChart继承于BarLineChartBase<CombinedData>，实现了CombinedDataProvider接口。持有CombinedChartRenderer(this, mAnimator, mViewPortHandler)的引用，CombinedData（多种数据）。

CombinedData（继承于BarLineScatterCandleBubbleData）持有LineData、BarData、ScatterData、CandleData、BubbleData的引用

触碰事件类

ChartTouchListener（抽象类），继承于GestureDetector.SimpleOnGestureListener，里面有一个枚举，声明了图表的动作状态NONE, DRAG, X\_ZOOM, Y\_ZOOM, PINCH\_ZOOM, ROTATE, SINGLE\_TAP, DOUBLE\_TAP, LONG\_PRESS, FLING。

BarLineChartTouchListener（继承于ChartTouchListener），在这个类中实现了GestureDetector.SimpleOnGestureListener的方法。

onTouch ()方法中，获取了VelocityTracker，回调

if (mTouchMode == NONE) {

mGestureDetector.onTouchEvent(event);

}

switch (event.getAction() & MotionEvent.ACTION\_MASK) {

case MotionEvent.ACTION\_DOWN://

break;

case MotionEvent.ACTION\_POINTER\_DOWN://多点触碰——按下

break;

case MotionEvent.ACTION\_MOVE:

break;

case MotionEvent.ACTION\_UP:

break;

case MotionEvent.ACTION\_POINTER\_UP://多点触碰——抬起

break;

}

控制统计图的扩大缩小（Zoom Out缩小，Zoom In 放大），以及触碰事件，显示高亮。

包含获取多点触碰时两个点之间的距离。

触碰事件监听 OnChartGestureListener。

Void onChartGestureStart()监听动作开始

void onChartGestureEnd()监听动作结束

onChartLongPressed(MotionEvent me)长按事件

onChartDoubleTapped(MotionEvent me)双击事件

void onChartSingleTapped(MotionEvent me) 单击事件

void onChartFling(MotionEvent me1, MotionEvent me2, float velocityX, float velocityY)惯性滑动

void onChartScale(MotionEvent me, float scaleX, float scaleY)放大缩小

void onChartTranslate(MotionEvent me, float dX, float dY)拖动

可以在此设置新的 功能

IvalueFormatter 这个接口实现了 Y 轴数据的格式化。

IaxisValueFormatter 这个接口实现了 X 轴数据的格式化。

LargeValueFormatter 拥有格式化数值很大的数据（比如12345格式化为12.3k）

辅助类：

ObjectPool< T extends ObjectPool.Poolable>，用于回收 object 实例，可复用

AbstractBuffer类：

提升画图的表现，原理：替代对象换为重建对象。

重要属性：

index，buffer（浮动缓冲区，用于保存要绘制的数据点，顺序：x、y、x、y，…），phaseX（x轴上的动画），phaseY（y轴上的动画），mFrom（x轴可见部分的起始点），mTo（x轴可见部分的结尾点）。

重要方法：

Feed(T data)使用提供的数据来填充buffer，当填充完毕，重置index属性。

ComponentBase（抽象）类，属性 mEnabled = true（是否使用，默认为 true），mXOffset，mYOffset，mTypeface， mTextSize， mTextColor。

AxisBase（抽象）类，继承于 ComponentBase，属性IAxisValueFormatter mAxisValueFormatter，用于格式化，mGridColor网格栏的颜色，mGridLineWidth 网格栏的宽度，mAxisLineColor 坐标轴的颜色，