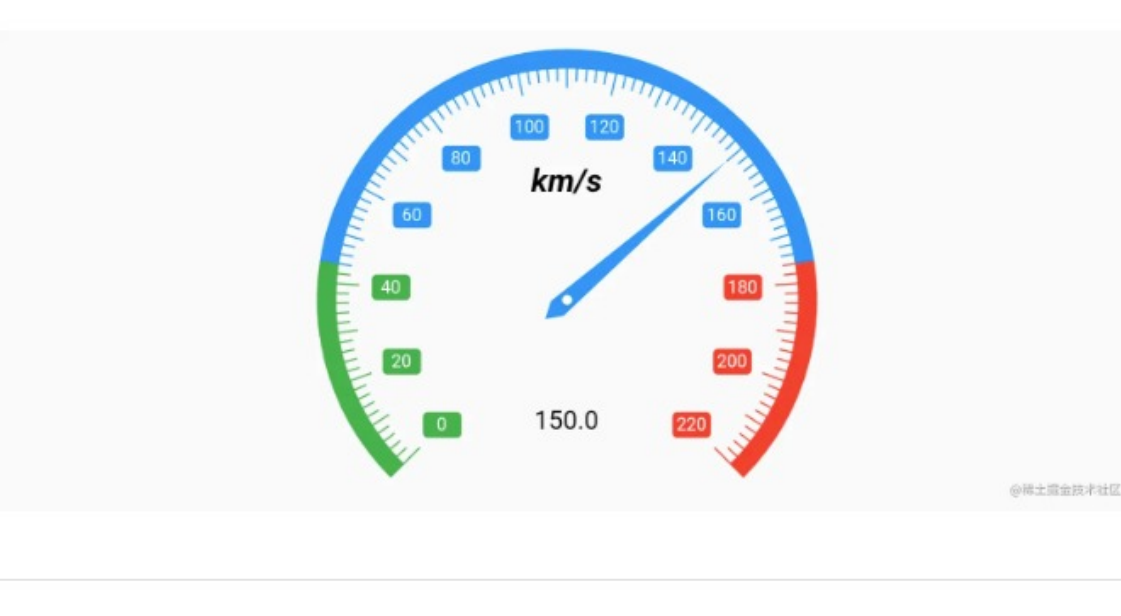
一、仪表图绘制

> 这节的目的是绘制出下面的仪表盘静态效果，通过绘制这个效果，你可以学到沿圆刻度的绘制 还有再旋转型的绘制中，如何保持文字方向的 水平 ，以及绘制多段色怎么实现。



1.常量与前期准备

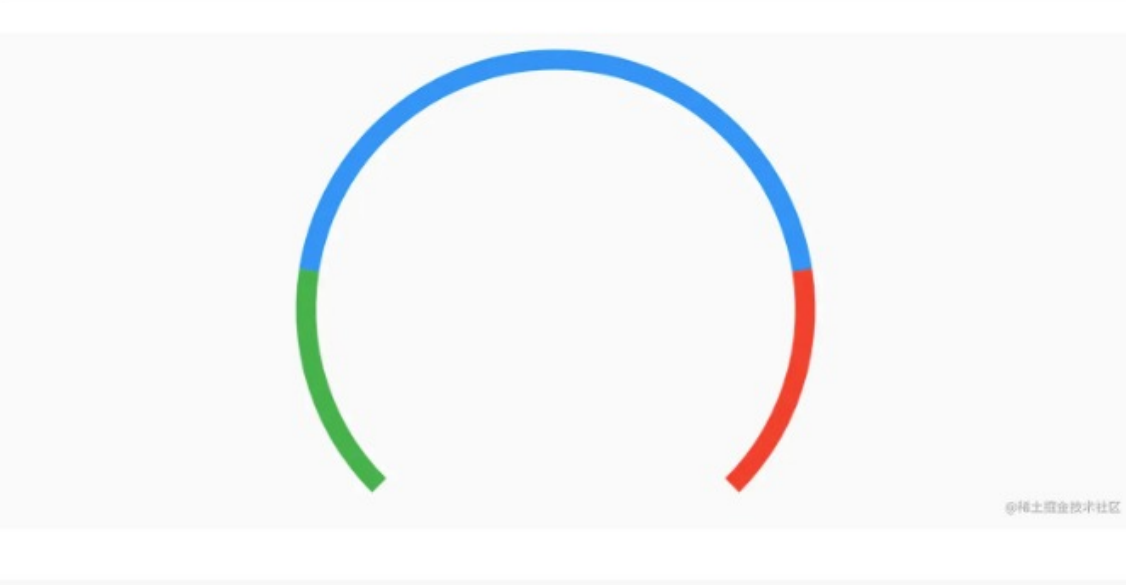
> 这里只实现静态效果，将一些量设置常量方便使用。你在之后可以对其完善，只需要将这些常量以参数的方式暴露给用户即可。私有常量和成员变量定义如下：

```dart
---->[p17/s06_panel/chart.dart]----
const double _kPiePadding = 20; // 圆边距
const double _kStrokeWidth = 12; // 圆弧宽
const double _kAngle = 270; // 圆弧角度
const int _kMax = 220;  // 最大刻度值
const int _kMin = 0; // 最小刻度值
const double _kScaleHeightLever1 = 14; // 短刻度线
const double _kScaleHeightLever2 = 18; // 逢5线
const double _kScaleHeightLever3 = 20; // 逢10线
const double _kScaleDensity = 0.3; // 逢10线
const double _kColorStopRate = 0.2; // 颜色变化分率
const List<Color> _kColors = [ // 颜色列表
  Colors.green,
  Colors.blue,
  Colors.red,
];

class ChartPainter extends CustomPainter {
  // 位置绘制器
  final TextPainter _textPainter = TextPainter(textDirection: TextDirection.ltr);
  double value = 150; //指针数值
  double radius = 0; // 圆半径

  Paint fillPaint = Paint(); // 填充画笔
  Paint stokePaint = Paint() ..strokeWidth = 1..style = PaintingStyle.stroke; // 线型画笔
```

---

2. 绘制三色圆弧

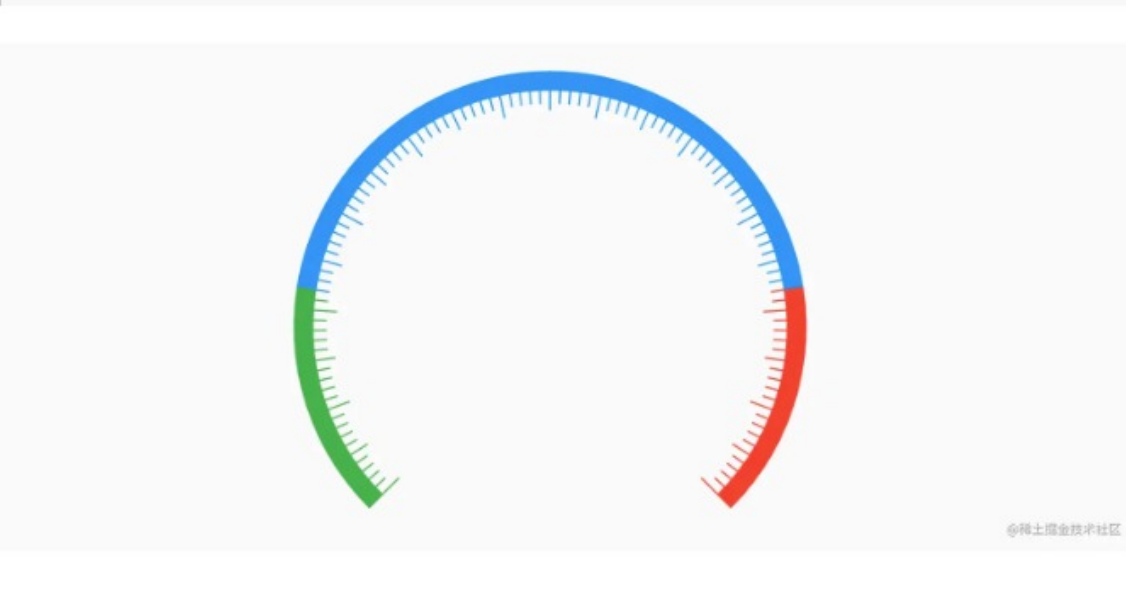
> 由于绘线和圆相关，将画布原点移至中心。并通过 `canvas.rotate(pi / 2);` 将旋转时的起点变为 y 轴正方向。在 `drawOutline` 中，从 `pi / 180 * initAngle` 开始画弧，也就是左下角，然后绘制 `pi / 180 * _kAngle` 也就是总弧度，称为一条路径。然后通过 路径测量 ，将路径分为三段绘制。通过 `_kColorStopRate` 控制小段占总长的分率。



```dart
@override
void paint(Canvas canvas, Size size) {
  radius = size.shortestSide / 2 - _kPiePadding; // 当前圆半径
  canvas.translate(size.width / 2, size.height / 2); // 画布原点移至中点
  canvas.rotate(pi / 2);
  drawOutline(canvas);
}

double get initAngle => (360 - _kAngle) / 2;

void drawOutline(Canvas canvas) {
  Path path = Path()..moveTo(radius, 0);
  path.arcTo(
      Rect.fromCenter( center: Offset.zero, width: radius * 2, height: radius * 2),
      pi / 180 * initAngle,
      pi / 180 * _kAngle,
      true);
  PathMetrics pms = path.computeMetrics();
  stokePaint..strokeWidth = _kStrokeWidth;
  // 通过路径测量绘制三段线
  pms.forEach((PathMetric pm) {
    canvas.drawPath(
        pm.extractPath(0, pm.length * _kColorStopRate), stokePaint..color = _kColors[0]);
    canvas.drawPath(pm.extractPath(pm.length * _kColorStopRate, pm.length * (1-_kColorStopRa
        stokePaint..color = _kColors[1]);
    canvas.drawPath(pm.extractPath(pm.length * (1-_kColorStopRate), pm.length),
        stokePaint..color = _kColors[2]);
  });
}
```

3.绘制刻度线

> 这个刻度的绘制和之前滑动篇的刻度尺类似，只不过当时是移动绘制，现在是旋转绘制。 注意点有：使用 `_kScaleDensity` 来设置格线的密度，如果为 1 ，则说明 min 到 max 之间，每个数都对应一个刻度，0.5 时，表示每隔两个数为 1 刻度，也就是格线的稀疏程度。

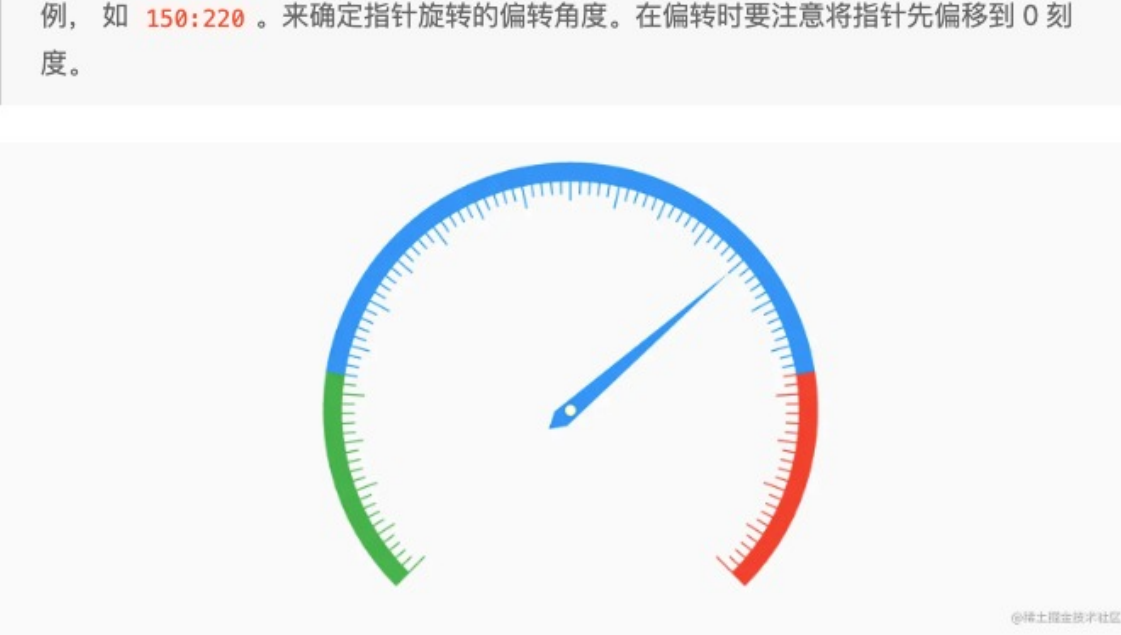


```dart
@override
void paint(Canvas canvas, Size size) {
  radius = size.shortestSide / 2 - _kPiePadding; // 当前圆半径
  canvas.translate(size.width / 2, size.height / 2); // 画布原点移至中点
  canvas.rotate(pi / 2); // 将起始点旋转到 y 轴正方向
  drawScale(canvas); // 绘制刻度
  drawOutline(canvas); // 绘制弧线
}

void drawScale(Canvas canvas) {
  canvas.save();
  canvas.rotate(pi / 180 * initAngle);
  double len = 0;
  Color color = Colors.red;
  var count = _kMax * _kScaleDensity; // 格线个数
  for (int i = _kMin; i <= count; i++) {
    if (i % 5 == 0 && i % 10 != 0) {
      len = _kScaleHeightLever2;
    } else if (i % 10 == 0) {
      len = _kScaleHeightLever3;
    } else {
      len = _kScaleHeightLever1;
    }
    if (i < count * _kColorStopRate) {
      color = Colors.green;
    } else if (i < count * (1 - _kColorStopRate)) {
      color = Colors.blue;
    } else {
      color = Colors.red;
    }
    canvas.drawLine(Offset(radius + _kStrokeWidth / 2, 0),
        Offset(radius - len, 0), stokePaint..color = color);
    canvas.rotate(pi / 180 / _kMax * _kAngle /_kScaleDensity);
  }
  canvas.restore();
}
```

---

4. 绘制指针

> 指针的绘制比较简单，需要在意的是当前偏转角度值的计算。使用当前值占最大值的比例， 如 `150:220` 。来确定指针旋转的偏转角度。在偏转时要注意将指针先偏移到 0 刻度。

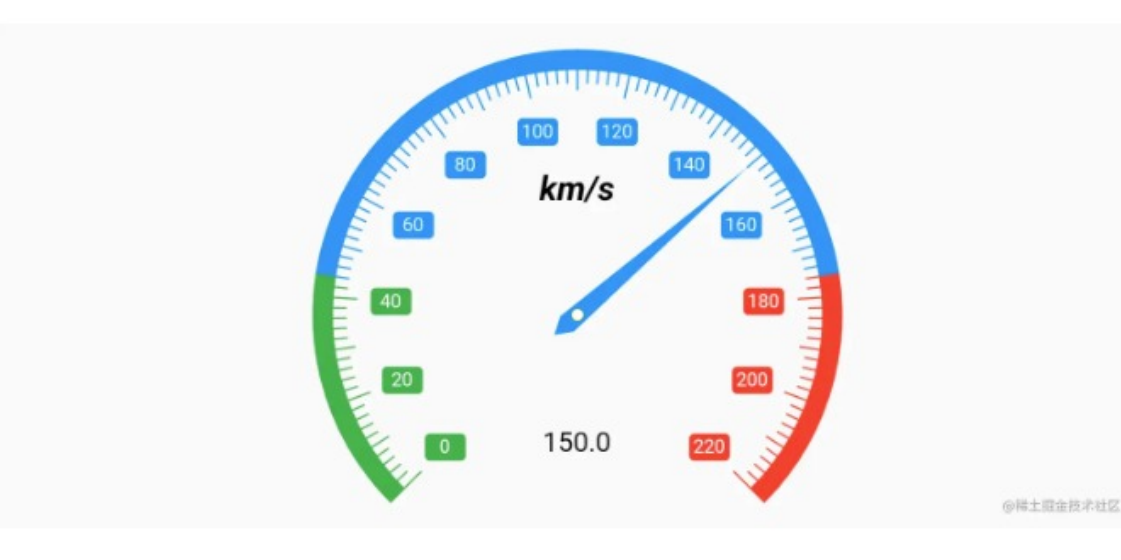


```dart
@override
void paint(Canvas canvas, Size size) {
  radius = size.shortestSide / 2 - _kPiePadding; // 当前圆半径
  canvas.translate(size.width / 2, size.height / 2); // 画布原点移至中点
  drawArrow(canvas); // 绘制指针
  canvas.rotate(pi / 2); // 将起始点旋转到 y 轴正方向
  drawScale(canvas); // 绘制刻度
  drawOutline(canvas); // 绘制弧线
}

void drawArrow(Canvas canvas) {
  var nowPer = value / _kMax;
  Color color = Colors.red;
  canvas.save();
  canvas.rotate(pi / 180 * (-_kAngle / 2 + nowPer * _kAngle));
  Path arrowPath = Path();
  arrowPath.moveTo(0, 18);
  arrowPath.relativeLineTo(-6, -10);
  arrowPath.relativeLineTo(6, -radius + 10);
  arrowPath.relativeLineTo(6, radius - 10);
  arrowPath.close();
  if (nowPer < _kColorStopRate) {
    color = _kColors[0];
  } else if (nowPer < (1 - _kColorStopRate)) {
    color = _kColors[1];
  } else {
    color = _kColors[2];
  }
  canvas.drawPath(arrowPath, fillPaint..color = color);
  canvas.drawCircle(Offset.zero, 3, stokePaint..color = Colors.yellow);
  canvas.drawCircle(Offset.zero, 3, fillPaint..color = Colors.white);
  canvas.restore();
}
```

---

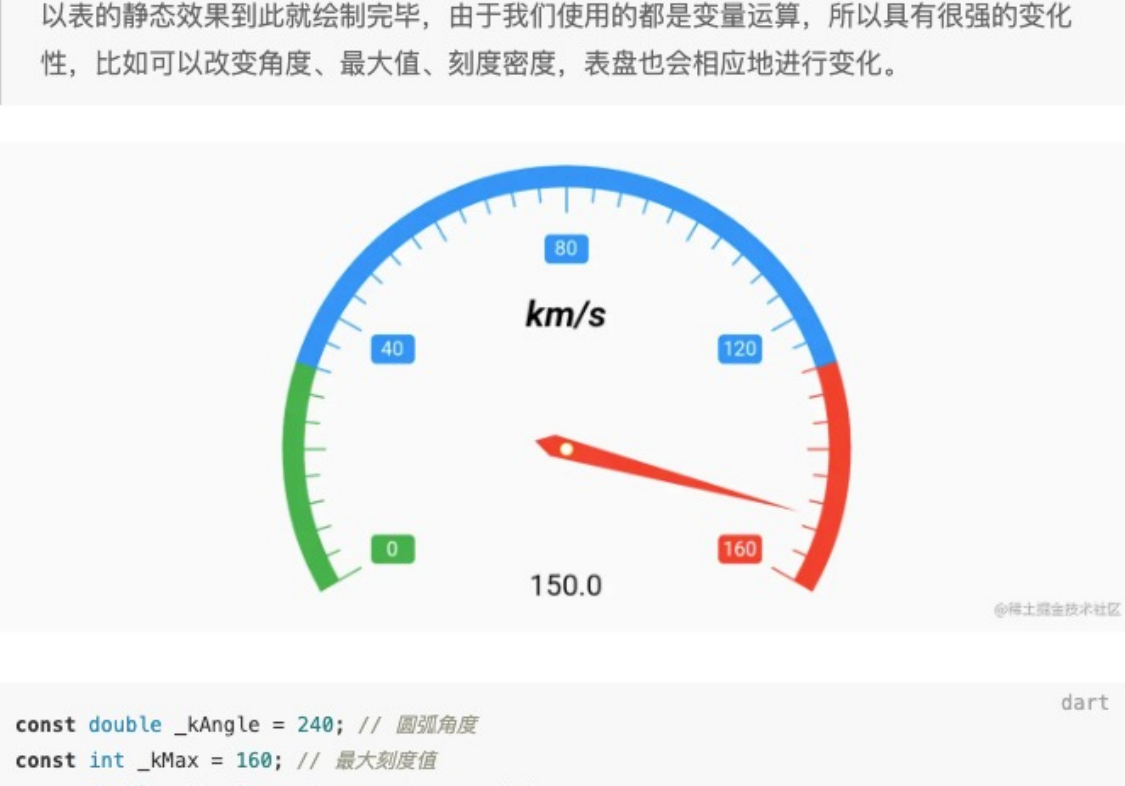
5. 绘制文字

> 这个可以说是最难的部分，我们需要根据刻度偏转角度，对文字进行定位，需要用到三角函数。



```dart
void drawText(Canvas canvas) {
  // 上方文字
  _drawAxisText(canvas, 'km/s',
      fontSize: 20,
      fontStyle: FontStyle.italic,
      fontWeight: FontWeight.bold,
      alignment: Alignment.center,
      color: Colors.black,
      offset: Offset(0, -radius / 2));
  // 下方文字
  _drawAxisText(canvas, '${value.toStringAsFixed(1)}',
      fontSize: 16,
      alignment: Alignment.center,
      color: Colors.black,
      offset: Offset(0, radius / 2));
  // 周围文字
  int count = (_kMax - _kMin)*_kScaleDensity ~/ 10;
  Color color = Colors.red;
  for (int i = _kMin; i <= count; i++) {
    var thta = pi / 180 * (90 + initAngle + (_kAngle / count) * i);
    if (i < count * _kColorStopRate) {
      color = _kColors[0];
    } else if (i < count * (1 - _kColorStopRate)) {
      color = _kColors[1];
    } else {
      color = _kColors[2];
    }
    Rect rect = Rect.fromLTWH((radius - 40) * cos(thta) - 12,
        (radius - 40) * sin(thta) - 8, 24, 16);
    canvas.drawRRect(RRect.fromRectAndRadius(rect, Radius.circular(3)),
        fillPaint..color = color);
    _drawAxisText(canvas, '${i * 10 ~/_kScaleDensity}',
        fontSize: 11,
        alignment: Alignment.center,
        color: Colors.white,
        offset: Offset((radius - 40) * cos(thta), (radius - 40) * sin(thta)));
  }
}
```

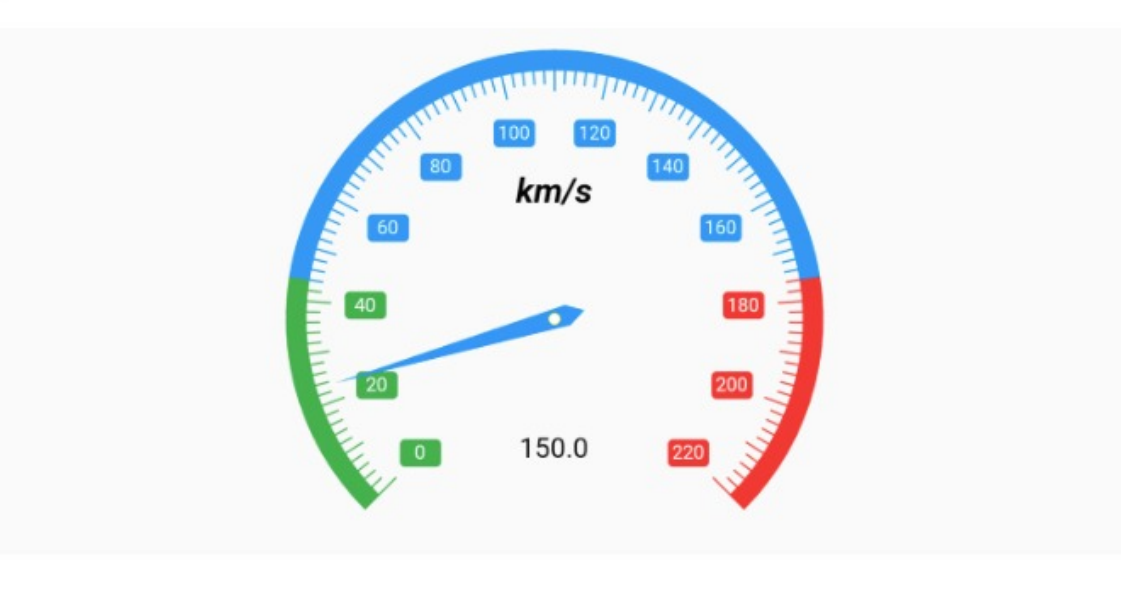
> 以表的静态效果到此就绘制完毕，由于我们使用的都是变量运算，所以具有很强的变化性，比如可以改变角度、最大值、刻度密度，表盘也会相应地进行变化。



```dart
const double _kAngle = 240; // 圆弧角度
const int _kMax = 160; // 最大刻度值
const double _kScaleDensity = 0.25; // 密度
const double _kScaleTextStep = 10; // 刻度文字步长
```

6.仪表盘指针动画

> 经过前面这么多的动画效果实现，指针转动的动画应该轻而易举了。核心就是在绘制指针时旋转画布操作里乘以动画器的值。



```dart
  void drawArrow(Canvas canvas) {
    var nowPer = value / _kMax;
    Color color = Colors.red;
    canvas.save();
    // 旋转时旋转值乘以动画器值
    double radians = pi / 180 * (-_kAngle / 2 + nowPer * _kAngle*repaint.value);
    canvas.rotate(radians);
```

二、绘制中国地图

> 如下，是通过 canvas 绘制的地图，是不是感觉 canvas 非常强大。只要有数据没有什么是画不出来的。 数据来源于 `https://datav.aliyun.com/tools/atlas`

---

1. 获取数据

> 通过 dio 请求接口，获取数据，传给绘制面板。其中数据对应的实体类 `MapRoot` 太长了，不贴在这，可详见源码。

```dart
---->[p17/s07_draw_map/map.dart]----
class ChinaMap extends StatefulWidget {
  @override
  _ChinaMapState createState() => _ChinaMapState();
}

class _ChinaMapState extends State<ChinaMap> {

  final String url =
      'https://geo.datav.aliyun.com/areas_v2/bound/100000_full.json'; //全国点位详细信息
  Future<MapRoot> _future;

 @override
  void initState() {
    super.initState();
    _future = getMapRoot();
  }

  //请求点位信息信息
  Future<MapRoot> getMapRoot() async {
    try {
      final Response response = await Dio().get(url);
      if (response.data != null) {
        return MapRoot.fromJson(response.data);
      } else {
        return null;
      }
    } catch (e) {
      return null;
    }
  }

  @override
  Widget build(BuildContext context) {
    return FutureBuilder<MapRoot>(
      future: _future,
      builder: (context, async) {
        if (async.hasData) {
          return  CustomPaint(
              size: Size(500, 400),
              painter: MapPainter(mapRoot: async.data),
          );
        } else {
          return CupertinoActivityIndicator();
        }
      },
    );
  }
}
```

2.绘制地图

> `mapRoot.features` 里装载的是省的数据，包括坐标和轮廓点信息，通过 `features.geometry.coordinates` 可以获取一个 四维 的数组，记录的是一个 三维 的点信息，可以通过这些点连成 Path 。这里特别要注意的是地图数据中 台湾省 、 海南省 和 九段线 没有下级区域，解析的方式和别的省份不同(这个点花费了我不少时间分析)。

```dart
class MapPainter extends CustomPainter {
  final MapRoot mapRoot; //点位信息

  Paint _paint;
  final List<Color> colors = [Colors.red, Colors.yellow, Colors.blue, Colors.green];
  int colorIndex = 0;

  MapPainter({this.mapRoot}) {
    _paint = Paint()
      ..strokeWidth = 0.1
      ..isAntiAlias = true;
  }

  @override
  void paint(Canvas canvas, Size size) {
    canvas.clipRect(
        Rect.fromPoints(Offset.zero, Offset(size.width, size.height)));
    canvas.translate(size.width / 2, size.height / 2);
    canvas.translate(-mapRoot.features[0].geometry.coordinates[0][0][0].dx,
        -mapRoot.features[0].geometry.coordinates[0][0][0].dy);
    canvas.translate(-560, 250);
    canvas.scale(6.5, -7.5);
    _drawMap(canvas, size);
  }

  void _drawMap(Canvas canvas, Size size) {
    //全国省份循环
    for (int i = 0; i < mapRoot.features.length; i++) {
      var features = mapRoot.features[i];
      PaintingStyle style;
      Color color;
      Path path = Path();
      if (features.properties.name == "台湾省" ||
          features.properties.name == "海南省" ||
          features.properties.name == "河北省" ||
          features.properties.name == "") { //海南和台湾和九段线
        features.geometry.coordinates.forEach((List<List<Offset>> lv3) {
          lv3.forEach((List<Offset> lv2) {
            path.moveTo(lv2[0].dx, lv2[0].dy);
            lv2.forEach((Offset lv1) {
              path.lineTo(lv1.dx, lv1.dy);
            });
          });
        });

        if (features.properties.name == "") {
          style = PaintingStyle.stroke;
          color = Colors.black;
        } else {
          style = PaintingStyle.stroke;
          color = colors[colorIndex % 4];
        }
        colorIndex++;
      } else {
        final Offset first = features.geometry.coordinates[0][0][0];
        path.moveTo(first.dx, first.dy);
        for (Offset d in features.geometry.coordinates.first.first) {
          path.lineTo(d.dx, d.dy);
        }
        style = PaintingStyle.stroke;
        color = colors[colorIndex % 4];
        colorIndex++;
      }
      canvas.drawPath(path, _paint..color = color..style = style); //绘制地图
    }
  }

  @override
  bool shouldRepaint(MapPainter oldDelegate) => oldDelegate.mapRoot != mapRoot;
}
```

> 可以通过改变画笔的填充来实现边框效果。

> 到这里，图表绘制的内容就结束了，下面将迎来本小册的最后一个篇章-- 粒子篇 。

留言

输入评论（Enter换行，⌘ + Enter发送）

发表评论

全部评论（9）

风二中 LV.1 JY.4 | Android&Flutter | 16天前

打卡

点赞 回复

混子中的混子 JY.3 | 前端混子 | 8月前

Path extractPath(
double start,
double end,
{bool startWithMoveTo = true}
)
...

展开

点赞 回复

_cloud LV.3 JY.3 | iOS @ HB | 1年前

你好，请问Flutter中能绘制立体图码？现在有个需求需要绘制3D柱状图，望指教，谢谢

点赞 回复

devLake LV.2 JY.3 | 全干工程师 | 1年前

要把表盘文字正过来好难啊 😭

点赞 1

> **白瑞德** | 1年前
>
> 可以先旋转定位置，再旋转回正角度
>
> 点赞 回复

andfaraway LV.2 JY.3 | iOS | Flutter 客户端 | 1年前

打个卡

点赞 回复

张江 LV.2 JY.4 | iOS高级工程师 | 2年前

封装发散一下

4 回复

煮一壶月光下酒 JY.1 | 酒徒 | 2年前

mark

点赞 回复

ZkJanus JY.3 | Android&Flutter @ axy | 2年前

踩一下！！！ 火钳刘明

点赞 回复