本节目标:

```dart
[1]. 认识 Dart 中的颜色表示方式。
[2]. 了解颜色 [混合模式] 的效果。
[3]. 了解如何读取图片中的像素颜色。
```

一、认识颜色

1.Dart 中的颜色

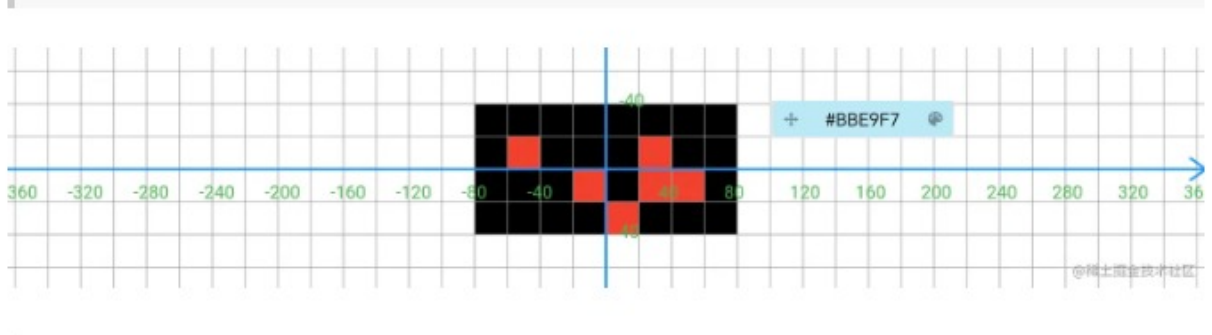
> `Color` 类在 `dart.ui` 包下, 在Dart里，颜色很简单，核心就是一个 颜色的 int 值。这代表 Color 类对应的是 四通道 32 位 的颜色。并没有提供RGBA颜色到HSV颜色互换的方法。 `Color` 类中主要有 6 个成员属性，核心是 int 类型的 value ，用来记录颜色信息。 通过 `get` 关键字 可以获取该颜色的 透明通道值 `alpha` 、 红色通道值 `red` 、 绿色通道值 `green` 、 蓝色通道值 `blue` 、 透明度 `opacity` 。

```dart
const Color(int value) : value = value & 0xFFFFFFFF;
  final int value;
  int get alpha => (0xff000000 & value) >> 24;
  double get opacity => alpha / 0xFF;
  int get red => (0x00ff0000 & value) >> 16;
  int get green => (0x0000ff00 & value) >> 8;
  int get blue => (0x000000ff & value) >> 0;
```

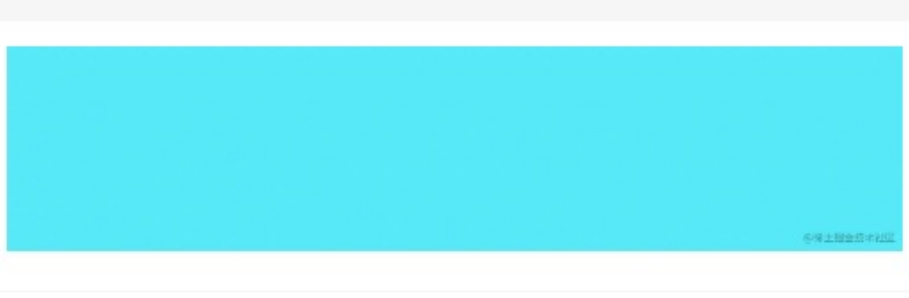
---

2. 关于颜色的 ARGB 表示

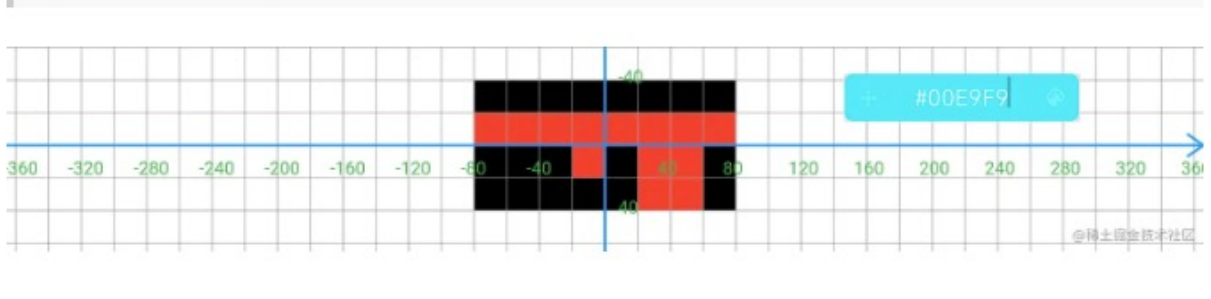
> 颜色的表示大家应该并不陌生。 比如 `0xffBBE9F7` 可以表示一个 32 位的颜色。它转化为二进制为 `11111111 10111011 111010011 1110111` ，每 8 个数字表示一个颜色通道，分别是 `ARGB` 。
> 颜色本质上是什么，这并不好说。毕竟 颜色本身是人类定义用来描述自然世界的工具 。我们能做的只是如何去 表示颜色 。其实 `0xffBBE9F7` 本身也只是一个代号而已，只是 人为规定 它和 `ARGB` 色彩空间 的一个颜色对应而已。
> 通过一个 int 值定义颜色，可以很方便的通过位运算对颜色进行变换。 `0xffBBE9F7` 可以转化为二进制，比如下面的示意图，黑色表示 1，红色表示 0。四行分别代表 `ARGB` 通道信息。



> 下面是用 `PhotoShop` 将 `0xffBBE9F7` 红色通道关闭前后的对比:

关闭红色通道前 `0xffBBE9F7`	关闭红色通道后 `0xff00E9F9`
	

> 从表现上来看，就是红色 R 通道 全部置 0 (即第二行全为红)，这和在程序中调用 `Color#.withRed(0)` 结果一致。下面是 源码中使用 `argb` 入参创建颜色 的代码。
> `withRed(int r)` 就是保持当前颜色对象的 AGB 量，用传入的值替换掉 R 。

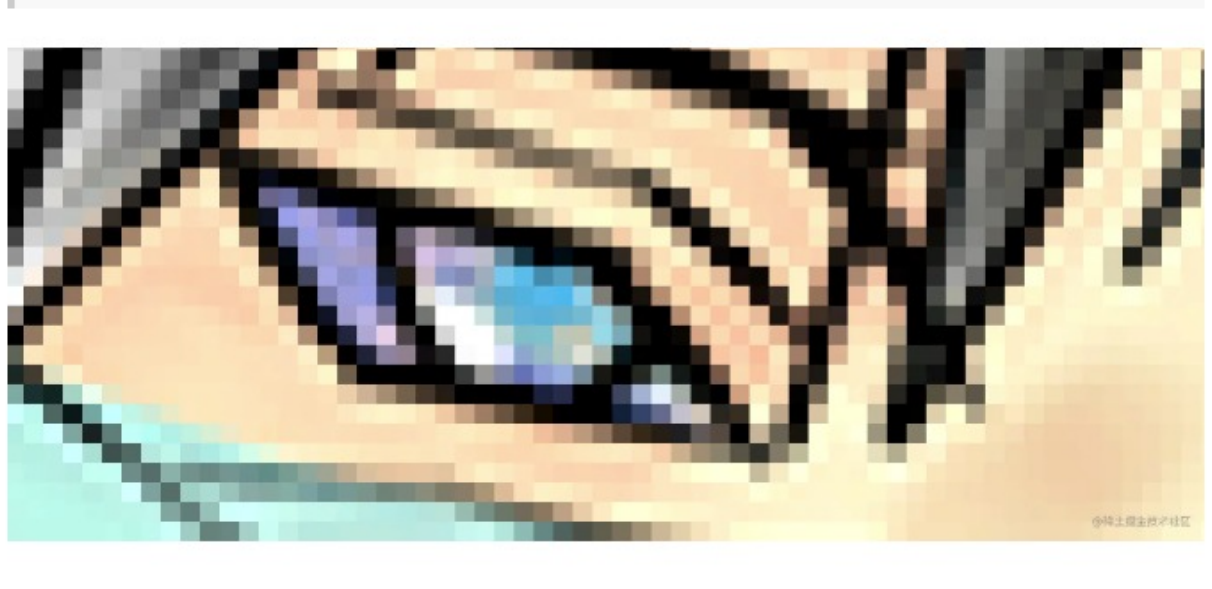


```dart
final Color colors = Color(0xffBBE9F7).withRed(0);

---->[ 源码中使用 argb 入参创建颜色 ]----
Color withRed(int r) {
  return Color.fromARGB(alpha, r, green, blue);
}

const Color.fromARGB(int a, int r, int g, int b) :
  value = (((a & 0xff) << 24) |
          ((r & 0xff) << 16) |
          ((g & 0xff) << 8)  |
          ((b & 0xff) << 0)) & 0xFFFFFFFF;
```

---

3. 关于颜色与图片

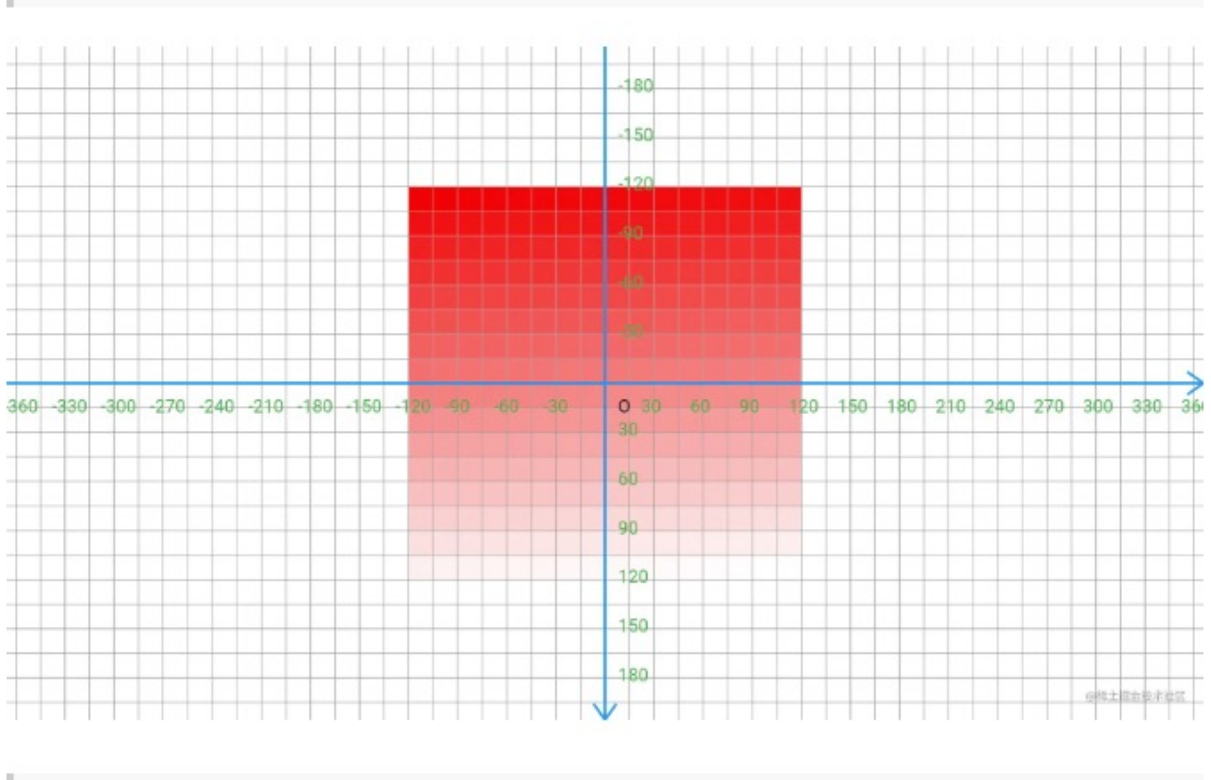
> 一张 位图图片 本质上是由一个个像素点组成的，每个像素都是一个颜色。也就是说位图是一个颜色集。我对图片的所有操作，本质上都是对颜色的变换。



> 对图片的滤色，如关闭红色通道，也就是让图片中所有的颜色 R 值置 0。

二 、绘制颜色阵列

> 下面是一个红色透明度依次降低的色块。是一个 `0xffff000~ox00ff0000` 共 256 个颜色的列表绘制而成图案。在这里想要介绍一下如何通过一维数组 取商 和 取余 绘制出类似二维(有行列)的效果。

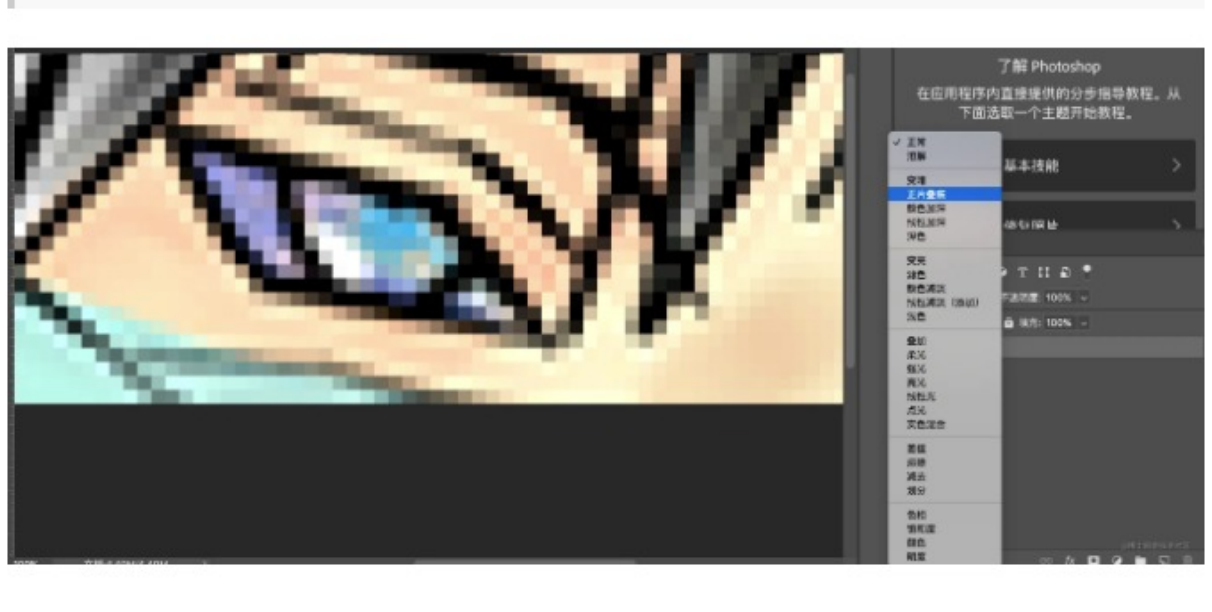


> 想象一下，在体育课上，我们先按从高到低排成一队，这时每人都有一个序号，比如我现在是 25 号。
> 然后老师让我们按顺序 8 个人一排，排成一个方阵，那么每个人的序号就变成了二维的行列。
> 那现在我的位置就在 `25 ~/ 8 = 3 行` ， `25 % 8 = 1 列` ，可用一个确定坐标 `(3,1)` 表示。
> 这是一个最简单的升维操作，比如上面就是 256 个元素每行 8 个排列出来的。

```dart
---->[p07_color/s02_colors/paper.dart]----
class PaperPainter extends CustomPainter {
  static const double step = 15; // 方格变长
  final Coordinate coordinate = Coordinate(step: step);
  // 颜色列表 256 个元素
  final List<Color> colors =
      List<Color>.generate(256, (i) => Color.fromARGB(255 - i, 255, 0, 0));

  @override
  void paint(Canvas canvas, Size size) {
    Paint paint = Paint();
    canvas.save();
    canvas.translate(size.width / 2, size.height / 2);
    // 遍历列表 绘制矩形色块
    canvas.translate(-step * 8.0, -step * 8.0);
    colors.asMap().forEach((i,color){
      int line = (i % 16); // 行
      int row = i ~/ 16; // 列
      var topLeft = Offset(step * line, step * row);
      var rect = Rect.fromPoints(topLeft, topLeft.translate(step, step));
      canvas.drawRect(rect, paint..color = color);
    });
    canvas.restore();

    coordinate.paint(canvas, size); //绘制坐标系
  }

  @override
  bool shouldRepaint(CustomPainter oldDelegate) => false;
}
```

三 、颜色的混合模式

> 混合模式在 `PS` 中的也有这个概念，表现如下。Dart 里一共有 29 种模式。下面将对这些模式进行测试，看一下它们的表现。



---

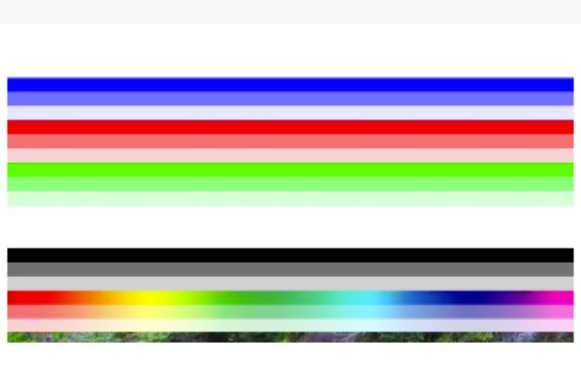
1. 了解混合模式的是什么

> 源码中注释为： `Algorithms to use when painting on the canvas.`
> 即 在画布上绘制时使用的算法 。当在画布上绘制 形状或图片 时，BlendMode决定混合像素时使用的算法。
> 既然是混合，那么肯定要两个东西混在一起，这就是 `src` 和 `dst` 。src 顾名思义是 源 ，dst 是 目标 。
> 一般 dst 会被认为是 背景 ，src 被认为是 正在绘制的图像 。 下面示例代码为红色叠合头像的效果。

```dart
---->[p07_color/s03_BlendMode/paper.dart]----
void paint(Canvas canvas, Size size) {
  if(image ==null) return;
  Paint srcPaint = Paint();
  canvas.translate(size.width / 2, size.height / 2);
  canvas.translate(-step * 17, -step * 7);
  Paint dstPaint = Paint();
  BlendMode.values.asMap().forEach((i, value) {
    int line = i ~/ 10;
    int row = i % 10;
    canvas.save();
    canvas.translate(3.7 * step * row, 5.5 * step * line);
    canvas.drawImageRect(image!, Rect.fromPoints(Offset.zero, Offset(image!.width*1.0,image
        Rect.fromCenter(center:Offset.zero, width: 25*2.0,height: 25*2.0), dstPaint);
    srcPaint
      ..color = Color(0xffff0000)
      ..blendMode = value;
    canvas.drawRect(
        Rect.fromPoints(Offset.zero, Offset(20 * 2.0, 20 * 2.0)), srcPaint);
    _simpleDrawText(
      canvas,value.toString().split(".")[1],
      offset: Offset(-10, 50));
    canvas.restore();
  });
}
```

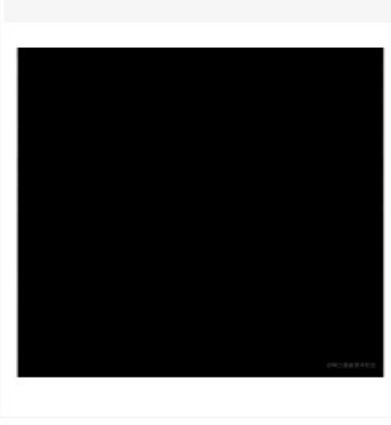
2.准备测试资源

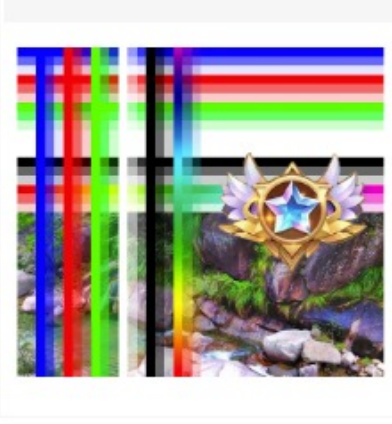
> 上面用一个颜色叠合图片，感觉示意并不是很明确。为此，我特意用PS 制作了两张官方示例中的图片，进行两个图片的叠合，更清晰的展示叠合效果。先说一下两张图的构成：
> dst： 背景图， 上方的彩条分别是 蓝、红、绿、白、黑、彩 。每色三个，分别是 `100%, 60%, 20%` 透明度。下方为一个 不透明图片 。
> src： 源图， 左的彩条分别是 蓝、红、绿、白、黑、彩 。每色三个，分别是 `100%, 60%, 20%` 透明度。右方为一个 背景透明的徽章 图片。

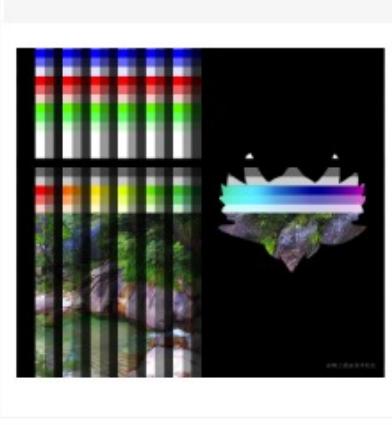
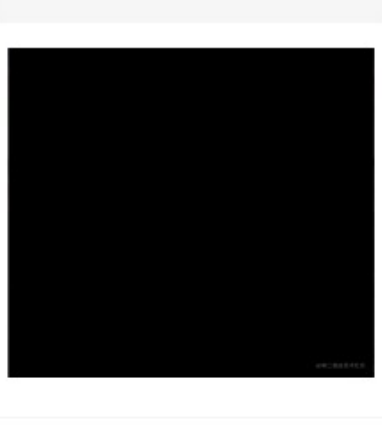
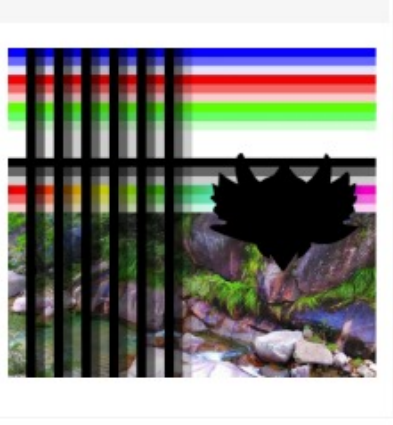
dst 背景	src 源
	

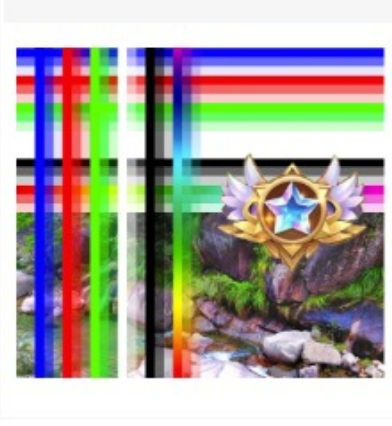
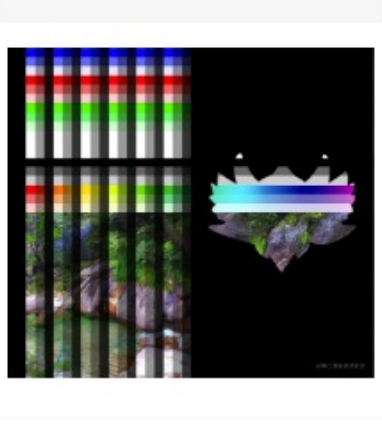
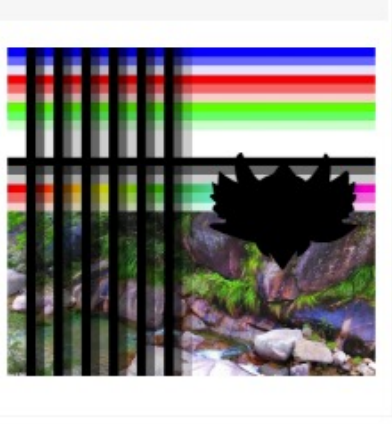
3.混合模式测试

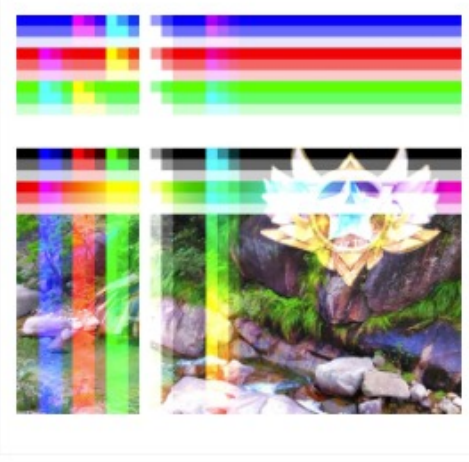
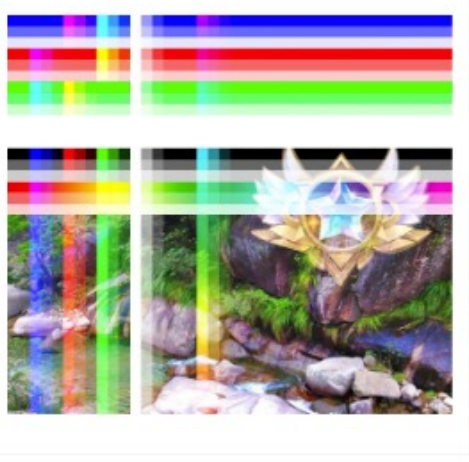
> 以下测试中，目标使用 srcOver ， 源图形使用 29 种不同的模式：
> 测试源码在: `p07_color/s04_BlendMode_detail`

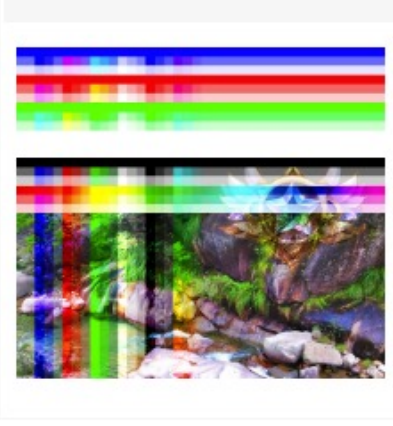
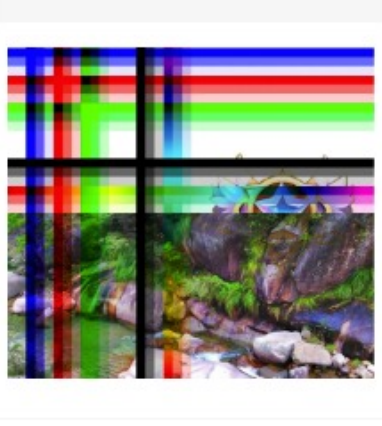
clear	src	dst
		

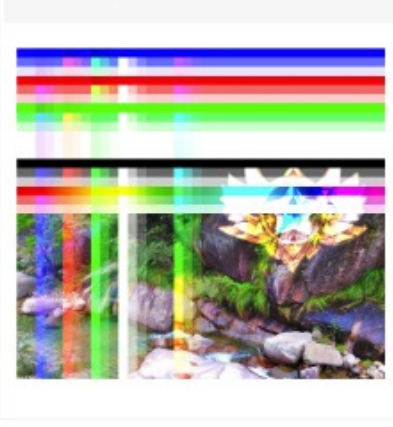
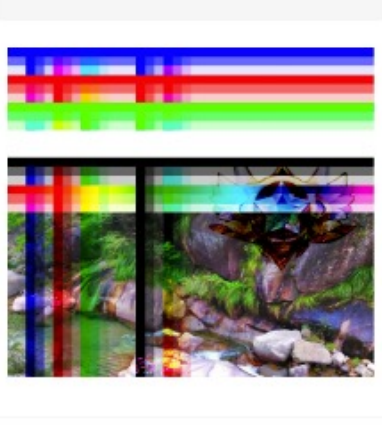
srcOver	dstOver	srcIn
		

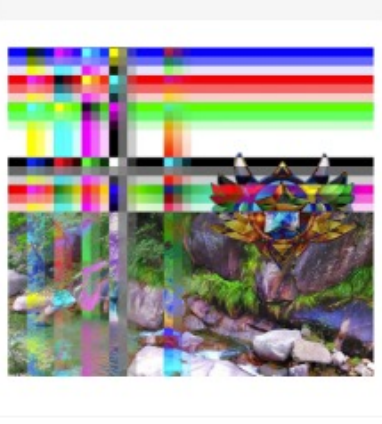
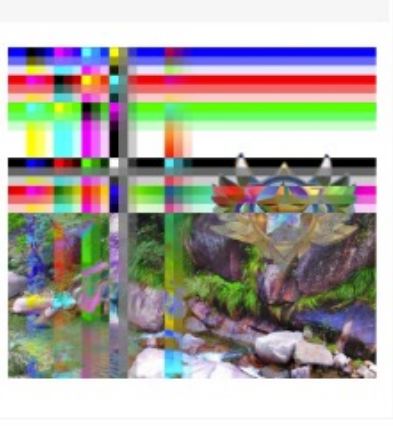
dstIn	srcOut	dstOut
		

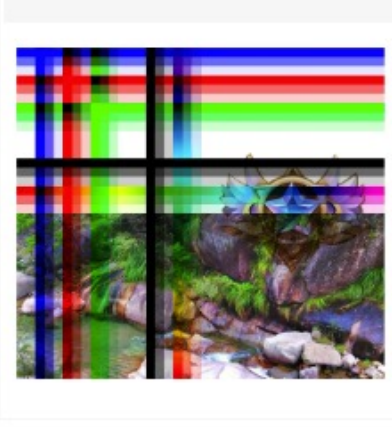
srcATop	dstATop	xor
		

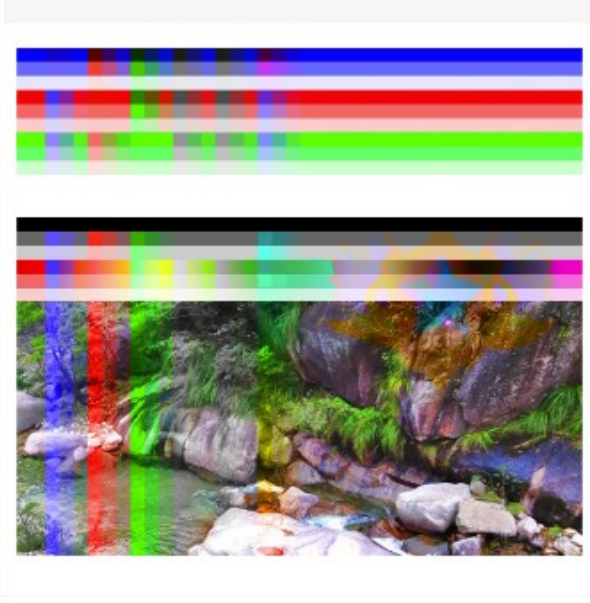
plus	modulate	screen
		

overlay	darken	lighten
		

colorDodge	colorBurn	hardLight
		

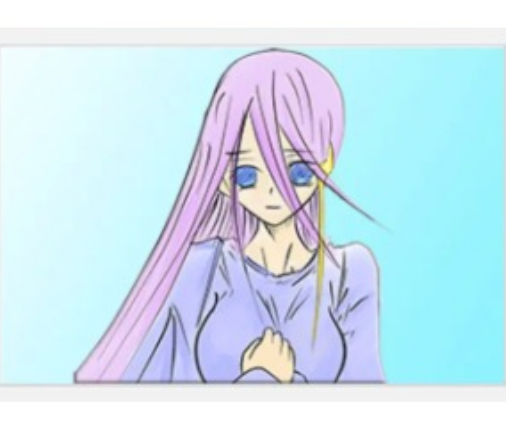
softLight	difference	exclusion
		

multiply	hue	saturation
		

color	luminosity
	

四、获取图片中的颜色

> 我们知道图片是由像素点构成的，每个像素点都 存有颜色信息 ， Flutter 本身的 Image 对象并没有获取像素的相关方法。 可以使用第三方 `image` 库进行相关操作。下面将使用这张图片进行演示：



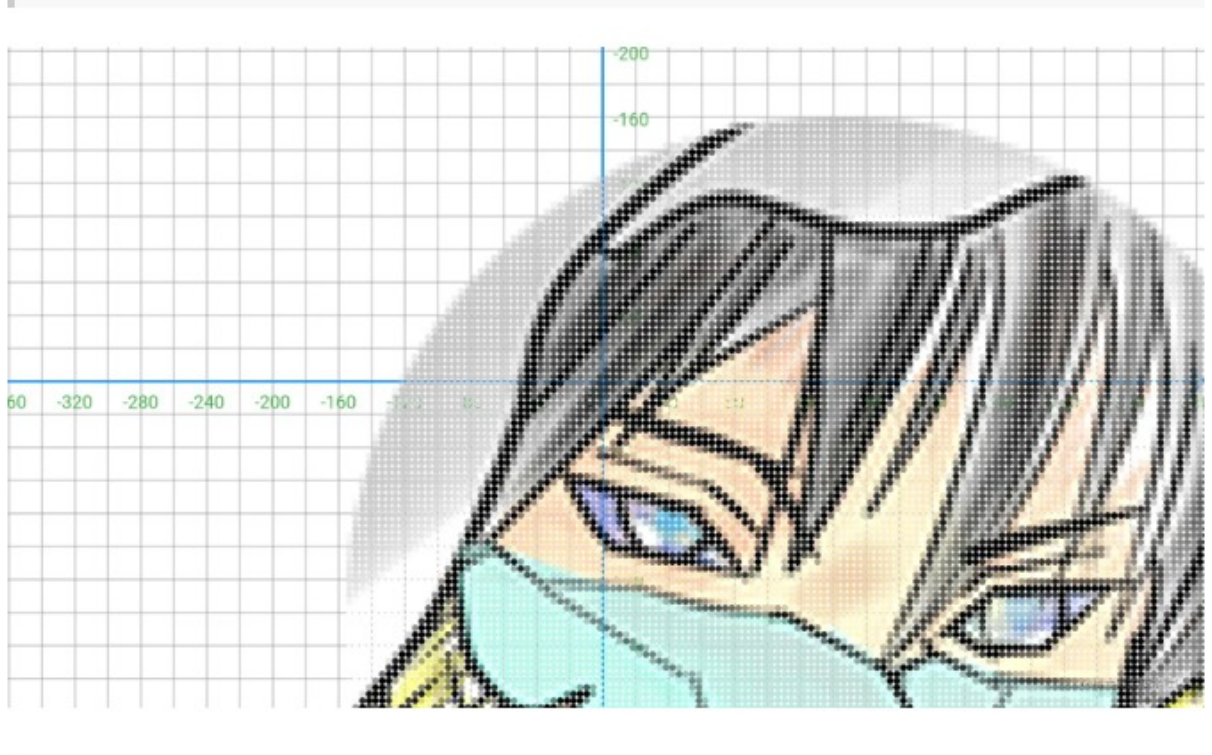
1.拾取图像的某一像素点颜色

> 如下，在画布上绘制出 图片中心点 的像素颜色。通过 `image` 包中的 `decodeImage` 方法可以将字节流转化为一个该包中的 `Image` 对象，这个对象可以通过 `getPixel` 方法获取获取某一坐标处的像素对象 Pixel 。

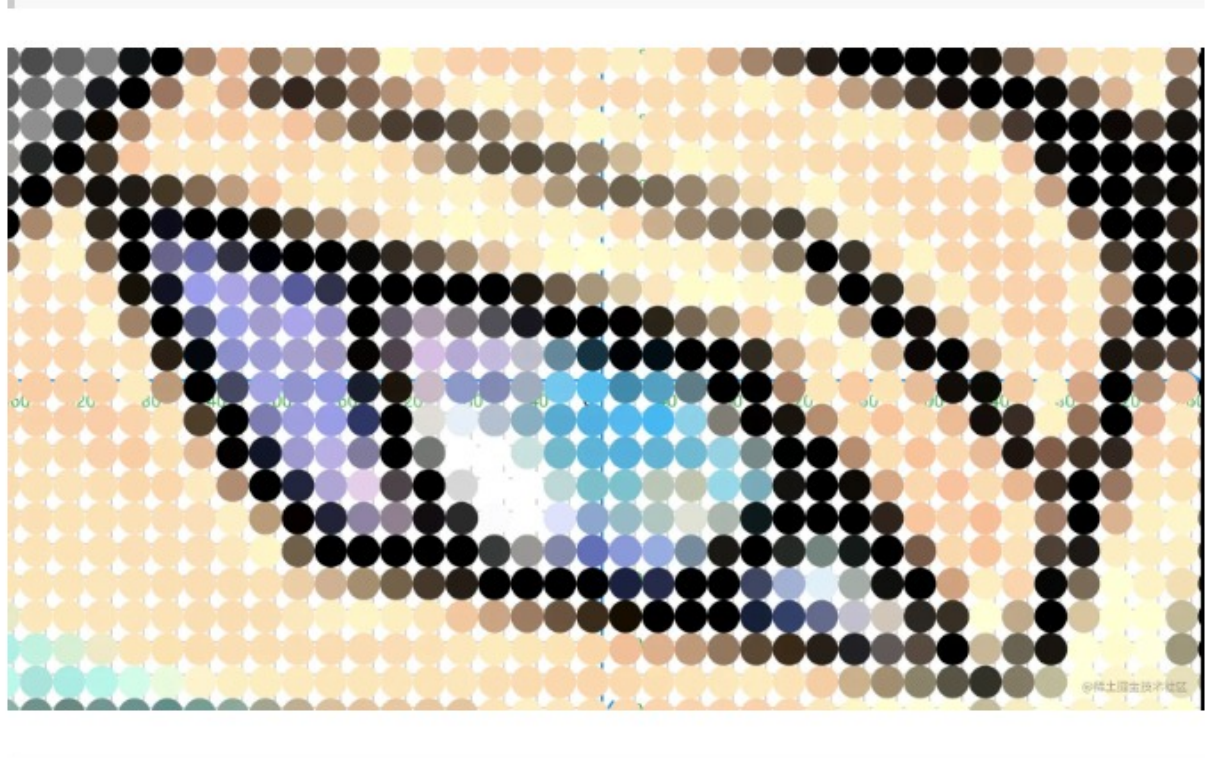
```dart
---->[p07_color/s05_image_color/paper.dart]----
import 'package:image/image.dart' as image;
//读取 assets 中的图片
Future<image.Image> loadImageFromAssets(String path) async {
  ByteData data = await rootBundle.load(path);
  List<int> bytes =
      data.buffer.asUint8List(data.offsetInBytes, data.lengthInBytes);
  return image.decodeImage(Uint8List.fromList(bytes));
}

---->[绘制方法]----
void _drawImage(Canvas canvas) {
  if (imageSrc == null)  return;
  image.Pixel pixel = imageSrc!.getPixel(imageSrc!.width, 0);
  var color = Color.fromARGB(pixel.a.toInt(),pixel.r.toInt(),pixel.g.toInt(),pixel.b.toInt(
  canvas.drawCircle(Offset.zero, 10, _paint..color = color);
}
```

2.绘制图片像素

> 通过获取图片的每个像素点，就可以 以自己的方式 绘制出图片。
> 如下，可以将图片按照一个个小圆绘制出来，当然你也可以用任意的形状。



> 你也可以通过小圆半径，控制图片绘制的大小。



> 对于每个位置的绘制信息使用一个 `Ball` 类进行维护。

```dart
---->[p07_color/s06_image_paint/paper.dart]----
class Ball {
  double x; //点位X
  double y; //点位Y
  Color color; //颜色
  double r; // 半径

  Ball({this.x=0, this.y=0, this.color=Colors.black, this.r=5});
}
```