本节目标

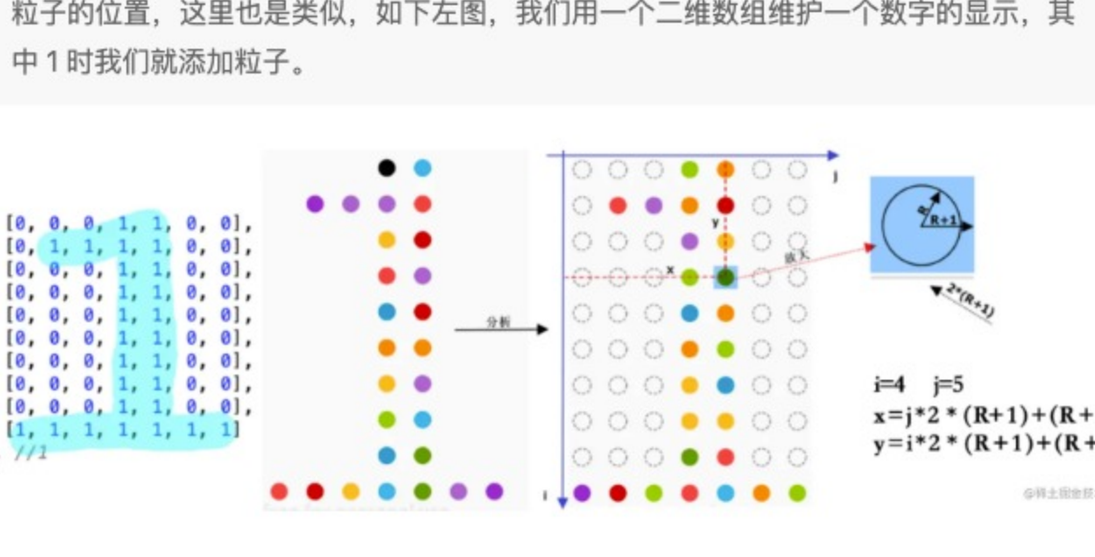
- 绘制如下粒子时钟

一、渲染粒子数字

> 首先来看一下如何渲染出如下的粒子的数字。

1.数字渲染思路

> 前面我们知道，对于粒子的东西，最重要的就是粒子的属性信息，尤其是位置。前面说过的使用图片资源进行文字的爆炸，实质上就是通过黑色像素在图片中的位置来映射出粒子的位置，这里也是类似，如下左图，我们用一个二维数组维护一个数字的显示，其中 1 时我们就添加粒子。



> 下面是显示出 `1994` 粒子的方法，通过 `collectParticles` 进行收集粒子。

```dart
collectParticles() {
  collectDigit(target: 1, index: 0);
  collectDigit(target: 9, index: 1);
  collectDigit(target: 9, index: 2);
  collectDigit(target: 4, index: 3);
}

void renderDigit({int target = 0, int index = 0}) {
  if (target > 10) {
    return;
  }
  double offSetX = 0;
  double space = _radius;
  offSetX = (digits[target][0].length * 2 * (_radius + 1) + space * 2) * index;
  for (int i = 0; i < digits[target].length; i++) {
    for (int j = 0; j < digits[target][j].length; j++) {
      if (digits[target][i][j] == 1) {
        double rX = j * 2 * (_radius + 1) + (_radius + 1); //第(i, j)个点圆心横坐标
        double rY = i * 2 * (_radius + 1) + (_radius + 1); //第(i, j)个点圆心纵坐标
        particles[count] = Particle(x: rX + offSetX, y: rY,
                                    size: _radius, color: Colors.blue);
        count++;
      }
    }
  }
}
```

2.粒子信息类和管理器

> 将粒子的属性信息为此成一个类。

```dart
class Particle {
  double x; /// x 位移.
  double y; /// y 位移.
  double size; /// 粒子大小.
  Color color; /// 粒子颜色.
  bool active; // 粒子是否可用
  double vx; /// 粒子水平速度.
  double ax; // 粒子水平加速度
  double ay; // 粒子竖直加速度
  double vy; ///粒子竖直速度.

  Particle({
    this.x = 0,
    this.y = 0,
    this.size = 0,
    this.vx=0,
    this.ax=0,
    this.ay=0,
    this.vy=0,
    this.active = true,
    this.color = Colors.black,
  });
}
```

> 粒子管理器中在构造方法里对粒子列表进行初始化，更新时根据索引对粒子信息进行修改即可，而不是频繁的添加或删除。每次 tick 方法被调用时，会根据时间来更新所有粒子信息，达到时钟的效果。

```dart
class ClockManage with ChangeNotifier {
  List<Particle> particles; /// 粒子列表
  int numParticles; /// 最大粒子数
  Size size; // 尺寸
  DateTime datetime; // 时间

  ClockManage({this.size, this.numParticles = 500}) {
    particles = List(numParticles);
    datetime = DateTime.now();
  }

  collectParticles() {
    collectDigit(target: 1, index: 0);
    collectDigit(target: 9, index: 1);
    collectDigit(target: 9, index: 2);
    collectDigit(target: 4, index: 3);
  }

  int count = 0;
  double _radius = 6;

  void collectDigit({int target = 0, int index = 0}) {
              // 同上
  }

  void tick(DateTime datetime) {
    collectParticles(datetime);
    notifyListeners();
  }
}
```

3.组件类和绘制类

> 在上一篇学会了使用 `Ticker` 来触发时间流，就消除了 AnimationController 对动画的逻辑束缚。初始化状态时，实例化 `ClockManage` 和 `Ticker`，并让粒子管理器中收集粒子。

```dart
class ClockPanel extends StatefulWidget {
  @override
  _ClockPanelState createState() => _ClockPanelState();
}

class _ClockPanelState extends State<ClockPanel> with SingleTickerProviderStateMixin {
  Ticker _ticker;
  ClockManage pm ;

  @override
  void initState() {
    super.initState();
    pm = ClockManage(size: Size(450, 300));
    pm.collectParticles();
    _ticker = createTicker(_tick)..start();
  }

  @override
  void dispose() {
    _ticker.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return  CustomPaint(
        size: pm.size,
        painter: ClockPainter(manage: pm),
    );
  }

  void _tick(Duration duration ) {
    if (DateTime.now().second % 1 == 0) pm.tick(DateTime.now());
  }
}
```

> 绘制很简单，遍历画圆即可，经过这几个类的协作，粒子的时钟便跃然纸上。

```dart
class ClockPainter extends CustomPainter {
  final ClockManage manage;

  ClockPainter({this.manage}) : super(repaint: manage);

  Paint clockPaint = Paint();

  @override
  void paint(Canvas canvas, Size size) {
    manage.particles.where((e) => e!=null).forEach((particle) {
      clockPaint..color = particle.color;
      canvas.drawCircle(
          Offset(particle.x, particle.y), particle.size, clockPaint);
    });
  }

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

二、粒子时间的绘制

> 前面虽然实现了粒子数字的绘制，但对于时间来说还差些小细节，比如 `:` 会影响粒子的定位。

1.粒子时间静态效果

> 通过一个自定义的偏移来校正位置，收集粒子后是如下效果。接下来我们只要解析时间，重设数字即可

```dart
collectParticles() {
  collectDigit(target: 0, offsetRate: 0);
  collectDigit(target: 3, offsetRate: 1);
  collectDigit(target: 10, offsetRate: 3.2);
  collectDigit(target: 2, offsetRate: 2.5);
  collectDigit(target: 8, offsetRate: 3.5);
  collectDigit(target: 10, offsetRate: 7.25);
  collectDigit(target: 4, offsetRate: 5);
  collectDigit(target: 9, offsetRate: 6);
}

void collectDigit({int target = 0, double offsetRate = 0}) {
  if (target > 10 && count>numParticles) {
    return;
  }
  double space = _radius* 2;
  double offSetX = (digits[target][0].length * 2 * (_radius + 1) + space) * offsetRate;
  for (int i = 0; i < digits[target].length; i++) {
    for (int j = 0; j < digits[target][j].length; j++) {
      if (digits[target][i][j] == 1) {
        double rX = j * 2 * (_radius + 1) + (_radius + 1); //第(i, j)个点圆心横坐标
        double rY = i * 2 * (_radius + 1) + (_radius + 1); //第(i, j)个点圆心纵坐标
        particles[count] = Particle(x: rX + offSetX, y: rY,
            size: _radius, color: Colors.green);
        count++;
      }
    }
  }
}
```

2.时间的数字解析

> 我们只需要将时间解析成数字即可，也非常简单。有一点要注意的是：我们未来避免对列表的频繁插入和删除操作，使用了定长的列表对粒子索引进行修改。这样会存在一点问题，比如 1 -> 2 时，只是标识了 2 对应的粒子，可能有些索引位是上次在 1 中的粒子，但不在 2 中，无法清除掉，可用在 `Particle` 中使用一个 `active` 标识是否绘制粒子来巧妙解决。在 collectParticles 时先把所有粒子的 active 标志设为 false，在修改时设置为 true。

```dart
collectParticles(DateTime datetime) {
  count = 0;
  particles.forEach((Particle element) {
    if(element!=null){
      element.active = false;
    }
  });
  collectDigit(target: datetime.hour ~/ 10, offsetRate: 0);
  collectDigit(target: datetime.hour % 10, offsetRate: 1);
  collectDigit(target: 10, offsetRate: 3.2);
  collectDigit(target: datetime.minute ~/ 10, offsetRate: 2.5);
  collectDigit(target: datetime.minute % 10, offsetRate: 3.5);
  collectDigit(target: 10, offsetRate: 7.25);
  collectDigit(target: datetime.second ~/ 10, offsetRate: 5);
  collectDigit(target: datetime.second % 10, offsetRate: 6);
}

void collectDigit({int target = 0, double offsetRate = 0}) {
  if (target > 10 && count > numParticles) {
    return;
  }

  double space = _radius * 2;
  double offSetX = (digits[target][0].length * 2 * (_radius + 1) + space) *
      offsetRate;
  for (int i = 0; i < digits[target].length; i++) {
    for (int j = 0; j < digits[target][j].length; j++) {
      if (digits[target][i][j] == 1) {
        double rX = j * 2 * (_radius + 1) + (_radius + 1); //第(i, j)个点圆心横坐标
        double rY = i * 2 * (_radius + 1) + (_radius + 1); //第(i, j)个点圆心纵坐标
        particles[count] = Particle(x: rX + offSetX,
            y: rY, size: _radius, color: Colors.green,
            active: true);
        count++;
      }
    }
  }
}
```

---

3. 时间的运动

> 为了尽可能少的减少绘制此时，并不需要要每次 Ticker 触发时 (16 ms) 都刷新。通过对比毫秒数时间戳是否大于 1000 即可。这样每秒进行一次粒子时钟的刷新，性能是最好的。通过这些操作，我们就实现了粒子时钟的基本效果。

```dart
class _ClockPanelState extends State<ClockPanel> with SingleTickerProviderStateMixin {
  Ticker _ticker;
  ClockManage pm ;

  @override
  void initState() {
    super.initState();
    pm = ClockManage(size: Size(450, 200));
    _ticker = createTicker(_tick)..start();
  }

  @override
  void dispose() {
    _ticker.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return  CustomPaint(
        size: pm.size,
        painter: ClockPainter(manage: pm),
    );
  }

  void _tick(Duration duration ) {
    if(DateTime.now().millisecondsSinceEpoch - pm.datetime.millisecondsSinceEpoch>1000){
      pm..datetime = DateTime.now()..tick();
    }
  }
}
```

三、时钟背景粒子

> 如下图，在时钟变化时，对应的数字会产生彩色粒子，进行运动。

> 由于背景粒子和粒子时钟的刷新时机不一样，可以使用两个画板，通过 Stack 进行堆叠。由于粒子需要不断进行运动，所以每次 `Ticker` 时都需要进行刷新。更新在 `checkParticles` 中检测粒子时钟变化的数字，进行添加可动的随机粒子。为避免粒子数量爆炸，可以在粒子运动到底端时移除粒子，最终可以将粒子的数量平均控制在 60 左右，这样对于渲染来说压力就不是很大了。

```dart
class BgManage with ChangeNotifier {
  List<Particle> particles;
  DateTime datetime; // 时间
  Random random = Random();

  /// 粒子列表
  int numParticles;

  /// 最大粒子数
  Size size; // 尺寸

  BgManage({this.size, this.numParticles = 500}) {
    particles = [];
    datetime = DateTime.now();
  }

  checkParticles(DateTime now) {
    //判断当前时间是否改变,再将点位放到集合中
    if ((datetime.hour ~/ 10) != (now.hour ~/ 10)) {
      collectDigit(target: datetime.hour ~/ 10, offsetRate: 0);
    }
    if ((datetime.hour % 10) != (now.hour % 10)) {
      collectDigit(target: datetime.hour % 10, offsetRate: 1);
    }
    if ((datetime.minute ~/ 10) != (now.minute ~/ 10)) {
      collectDigit(target: datetime.minute ~/ 10, offsetRate: 2.5);
    }
    if ((datetime.minute % 10) != (now.minute % 10)) {
      collectDigit(target: datetime.minute % 10, offsetRate: 3.5);
    }
    if ((datetime.second ~/ 10) != (now.second ~/ 10)) {
      collectDigit(target: datetime.second ~/ 10, offsetRate: 5);
    }
    if ((datetime.second % 10) != (now.second % 10)) {
      collectDigit(target: datetime.second % 10, offsetRate: 6);
      datetime = now;
    }
  }

  double _radius = 4;

  void collectDigit({int target = 0, double offsetRate = 0}) {
    if (target > 10) {
      return;
    }
    double space = _radius * 2;
    double offSetX =
        (digits[target][0].length * 2 * (_radius + 1) + space) * offsetRate;

    for (int i = 0; i < digits[target].length; i++) {
      for (int j = 0; j < digits[target][j].length; j++) {
        if (digits[target][i][j] == 1) {
          double rX = j * 2 * (_radius + 1) + (_radius + 1); //第(i, j)个点圆心横坐标
          double rY = i * 2 * (_radius + 1) + (_radius + 1); //第(i, j)个点圆心纵坐标
          Particle particle = Particle(
              x: rX + offSetX,
              y: rY,
              size: _radius,
              color: randomColor(),
              active: true,
              vx: 2.5 * random.nextDouble() * pow(-1, random.nextInt(20)),
              vy: 2 * random.nextDouble() + 1);
          particles.add(particle);
        }
      }
    }
  }

  Color randomColor({
    int limitA = 120,
    int limitR = 0,
    int limitG = 0,
    int limitB = 0,
  }) {
    var a = limitA + random.nextInt(256 - limitA); //透明度值
    var r = limitR + random.nextInt(256 - limitR); //红值
    var g = limitG + random.nextInt(256 - limitG); //绿值
    var b = limitB + random.nextInt(256 - limitB); //蓝值
    return Color.fromARGB(a, r, g, b); //生成argb模式的颜色
  }

  void tick(DateTime now) {
    checkParticles(now);

    for (int i = 0; i < particles.length; i++) {
      doUpdate(particles[i]);
    }
    notifyListeners();
  }

  void doUpdate(Particle p) {
    p.vy += p.ay; // y加速度变化
    p.vx += p.ax; // x加速度变化
    p.x += p.vx;
    p.y += p.vy;

    if (p.x > size.width) {
      p.x = size.width;
      p.vx = -p.vx;
    }

    if (p.x < 0) {
      p.x = 0;
      p.vx = -p.vx;
    }

    if (p.y > size.height) {
      particles.remove(p);
    }
  }
}
```

> 其他逻辑基本没什么需要特别指出的，和上面基本一致，可以参见源码。这里的粒子并不仅是圆而已，你可以换成任意可以绘制的东西。到这里这本小册就完结散花了，谢谢你的支持～ 疙疙江湖路修远，丹心相伴自有期。 我是 张风捷特烈 ，咱们有缘再会。

留言

输入评论（Enter换行，⌘ + Enter发送）

发表评论

全部评论（9）

Fssj · 3月前
particles[count] = Particle(
x: rX + offSetX,
y: rY,
size: _radius,
color: Colors.red,
active: true);...
展开
点赞 回复

iChunzhen · 1年前
为什么按钮点击会出发customPainter 的paint方法
点赞 回复

ZeroFlutter · Flutter 高级工程师 @ Fl... · 1年前
江湖再见
点赞 回复

张江 · iOS高级工程师 · 2年前
小册已阅读，茫茫江湖路 有缘再会
1 回复

天色将变 · app开发 · 2年前
好
点赞 回复

煮一壶月光下酒 · 逍徒 · 2年前
mark，已签收
点赞 回复

当前日期 · iOS @ 稀土掘金 · 2年前
大佬, 绘制复杂对象的时候, 如何避免阻塞主线程呢?用了compute, 但是貌似获取不到Canvas对象...
点赞 回复

Symboll · 前端工程师 · 2年前
这个时钟 跟 慕课网上bobobo老师那个canvas 一样啊。
点赞 1

张风捷特烈（作者） · 2年前
嗯，他是在 html5 里画的，我复刻在Android、Flutter 上了。
点赞 回复