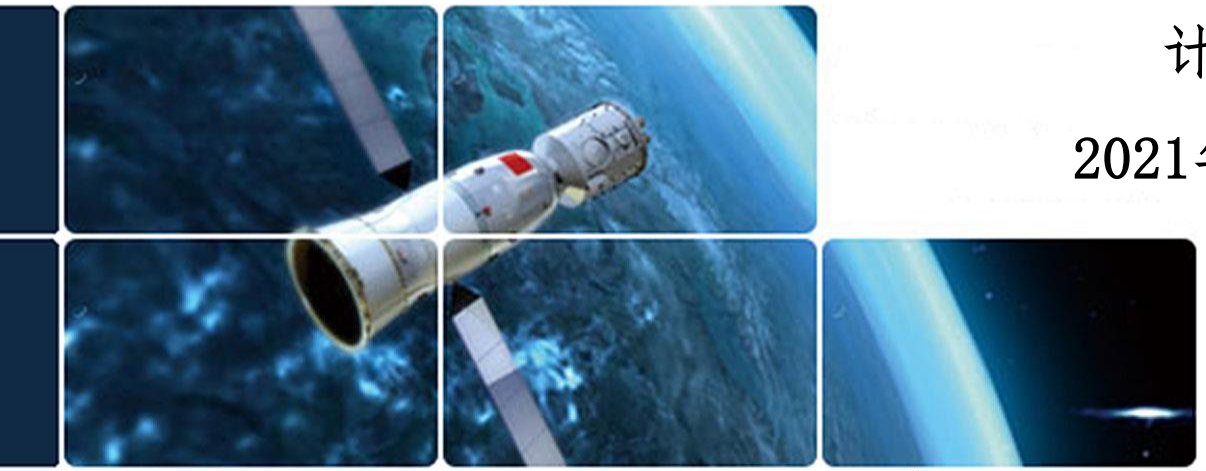
# 第6-2讲 移动互联网应用

# 图形和动画技术



计算学部

2021年11月29日

# 主要内容

- ***Harmony OS JS UI*框架**
- 绘制图形的画布
- 图形绘制
- 图像处理
- 动画基础
- *OpenGL ES*基础

# JS UI框架

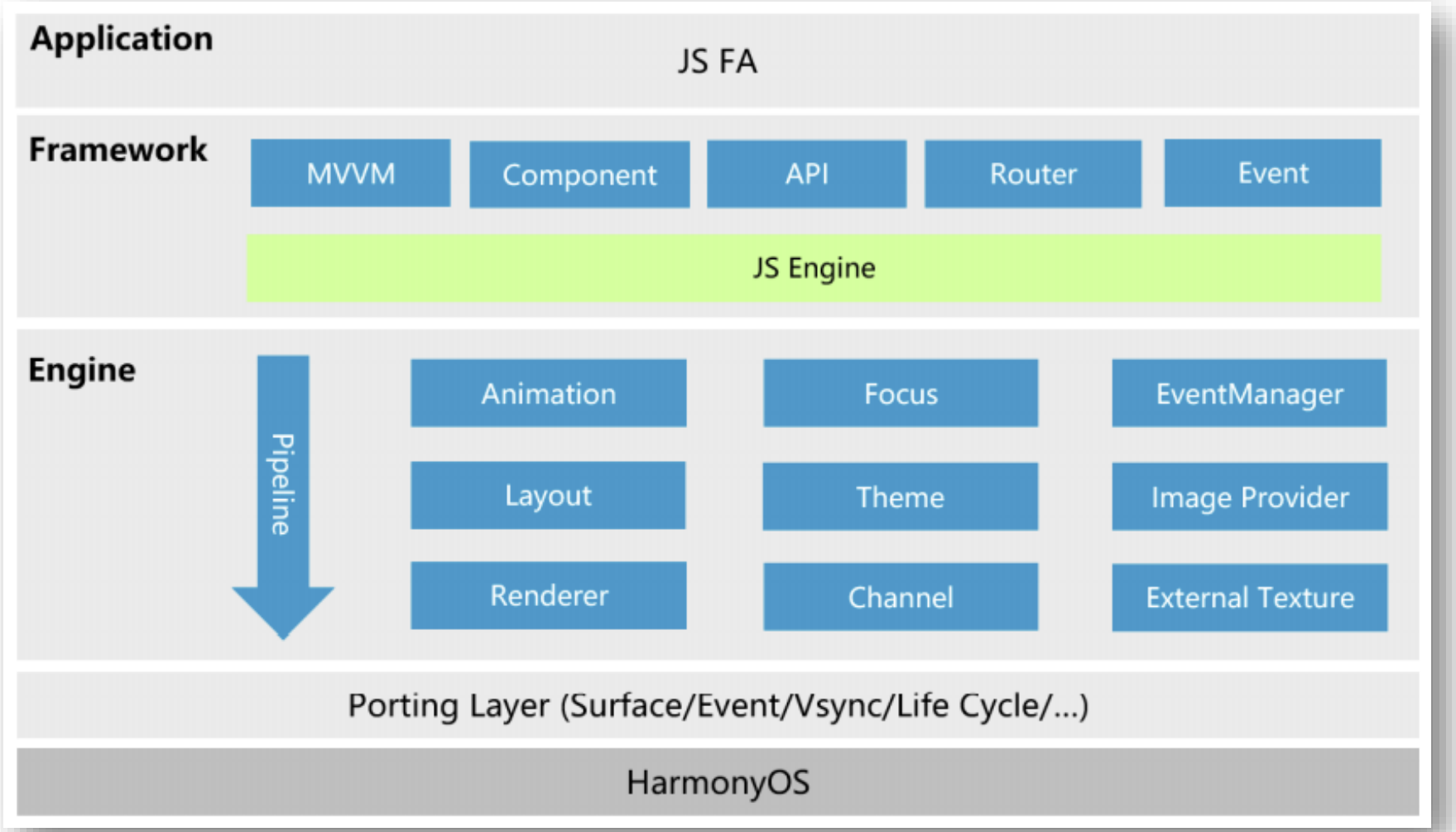
- *JS UI* 框架是一种跨设备*UI*开发框架，支持声明式编程和跨设备多态*UI*，编程工具包括：*HTML5*、*CSS* 和 *JavaScript*

- *JS UI* 框架采用 类*HTML* 和 *CSS* 声明式编程语言作为页面布局和页面样式的开发语言，页面业务逻辑支持*ECMAScript*规范的*JavaScript*语言。*JS UI* 框架声明式编程可以让开发者避免编写 *UI* 状态切换的代码，视图配置信息更直观

- *JS UI* 框架架构上支持*UI*跨设备显示能力，运行时自动映射到不同设备类型，开发者无感知，降低开发者多设备适配成本

- *JS UI* 框架包含了许多核心*UI*控件，如列表、图片和各类容器组件等，针对声明式语法进行了渲染流程的优化

- *JS UI*框架包括
  (1) ***Application***
  (2) ***Framework***
  (3) ***Engine***
  (4) ***Porting Layer***



# 初步体验JS FA应用

- *JS UI*框架支持纯*JavaScript*、*JavaScript*和*Java*混合语言开发
- *JS FA*指基于*JavaScript*或*JavaScript*和*Java*混合开发的*FA*，*AceAbility*是*JS FA*在*HarmonyOS*上运行时需要的基类，*AceAbility*是*Ability*的子类，应用运行入口类从*AceAbility*类派生

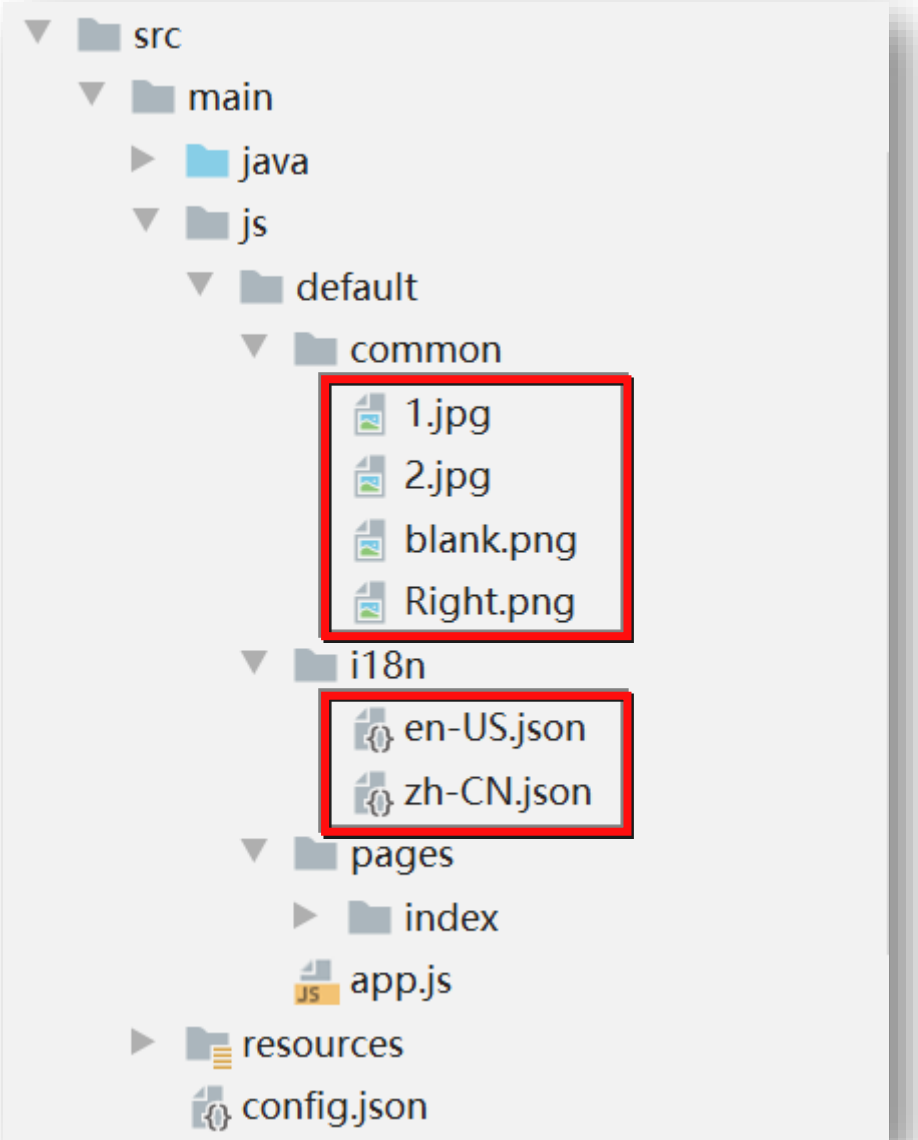
```
1. public class MainAbility extends AceAbility {
2.     @Override
3.     public void onStart( Intent intent) {
4.         super.onStart(intent);
5.     }
6.
7.    @Override
8.    public void onStop() {
9.        super.onStop();
10.   }
11. }
```

# 加载JS FA

- *JS FA*生命周期事件分为应用生命周期和页面生命周期，应用通过*AceAbility*类的*setInstanceName()*接口设置*Ability*的实例资源，并通过*AceAbility*窗口显示，对全局应用生命周期进行管理
- *setInstanceName(String name)*中参数*name*指实例名称，与*config.json*文件中*profile.application.js.name*对应。（1）若使用*default*，无需调用此接口；（2）若修改了实例名，则需在*Ability*实例的*onStart()*中调用此接口，*name*设置为新实例名，在*MainAbility*的*onStart()*的*super.onStart*()前调用此接口

```
1. public class MainAbility extends AceAbility {
2.     @Override
3.     public void onStart( Intent intent) {
           // config.json 配置文件中 ability.js.name 的标签值。
4.         setInstanceName("JSComponentName");
5.         super.onStart(intent);
6.     }
7. }
```

```
"js": [
  {
    "pages": [
      "pages/index/index"
    ],
    "name": "default",
    "window": {
      "designWidth": 454,
      "autoDesignWidth": true
    }
  }
]
```



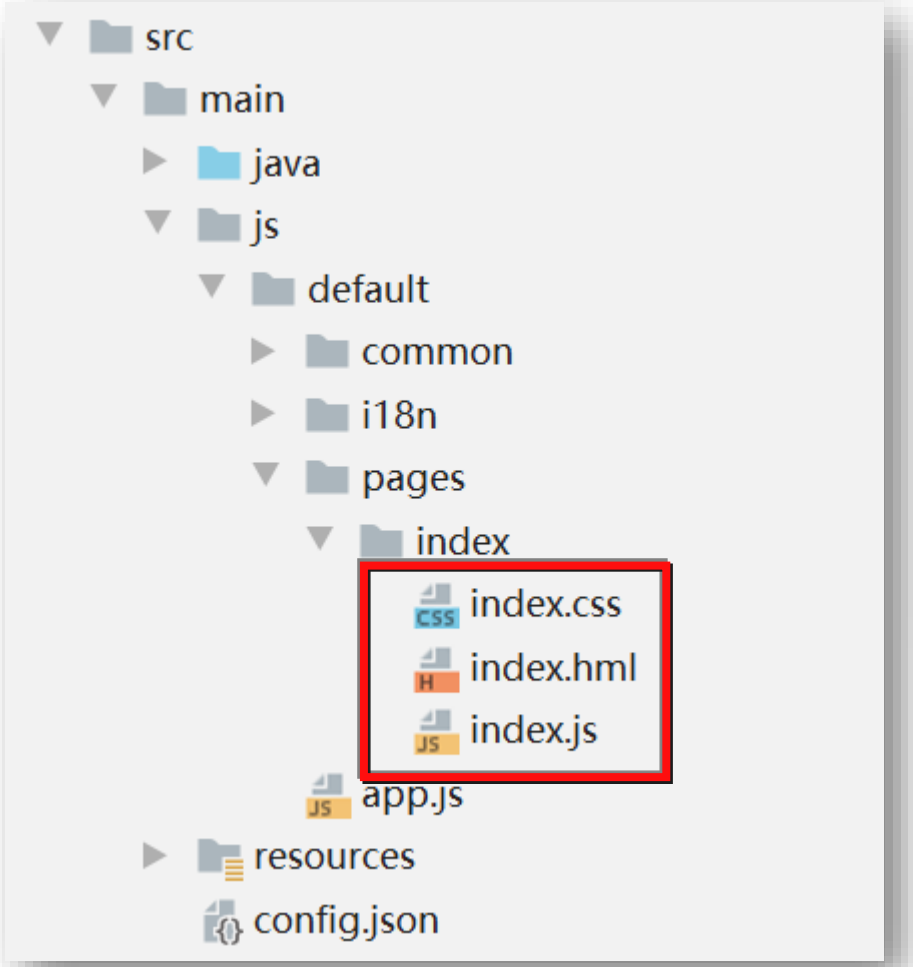

- ***common***文件夹主要存放公共资源，如图片、视频等
- ***i18n***文件夹存放多语言***json***文件：
  （1）***en-US.json*** 文件定义了英文模式下页面显示的变量
  （2）***zh-CN.json*** 文件定义了中文模式下页面显示的变量

```
{
  "strings": {
    "title": "Title",
    "subtitle": "Subtitle",
    "list": "List"
  },
  "Files": {
  }
}
```

```
{
  "strings": {
    "title": "标题",
    "subtitle": "副标题",
    "list": "栏目"
  },
  "Files": {
  }
}
```

# JS FA开发目录

- *pages*文件夹存放多个页面，每个页面由*hml*、*css*和*js*文件组成




index.css

```css
.container {
    flex-direction: column;
    justify-content: center;
    align-items: center;
}

.title {
    font-size: 80px;
}
```

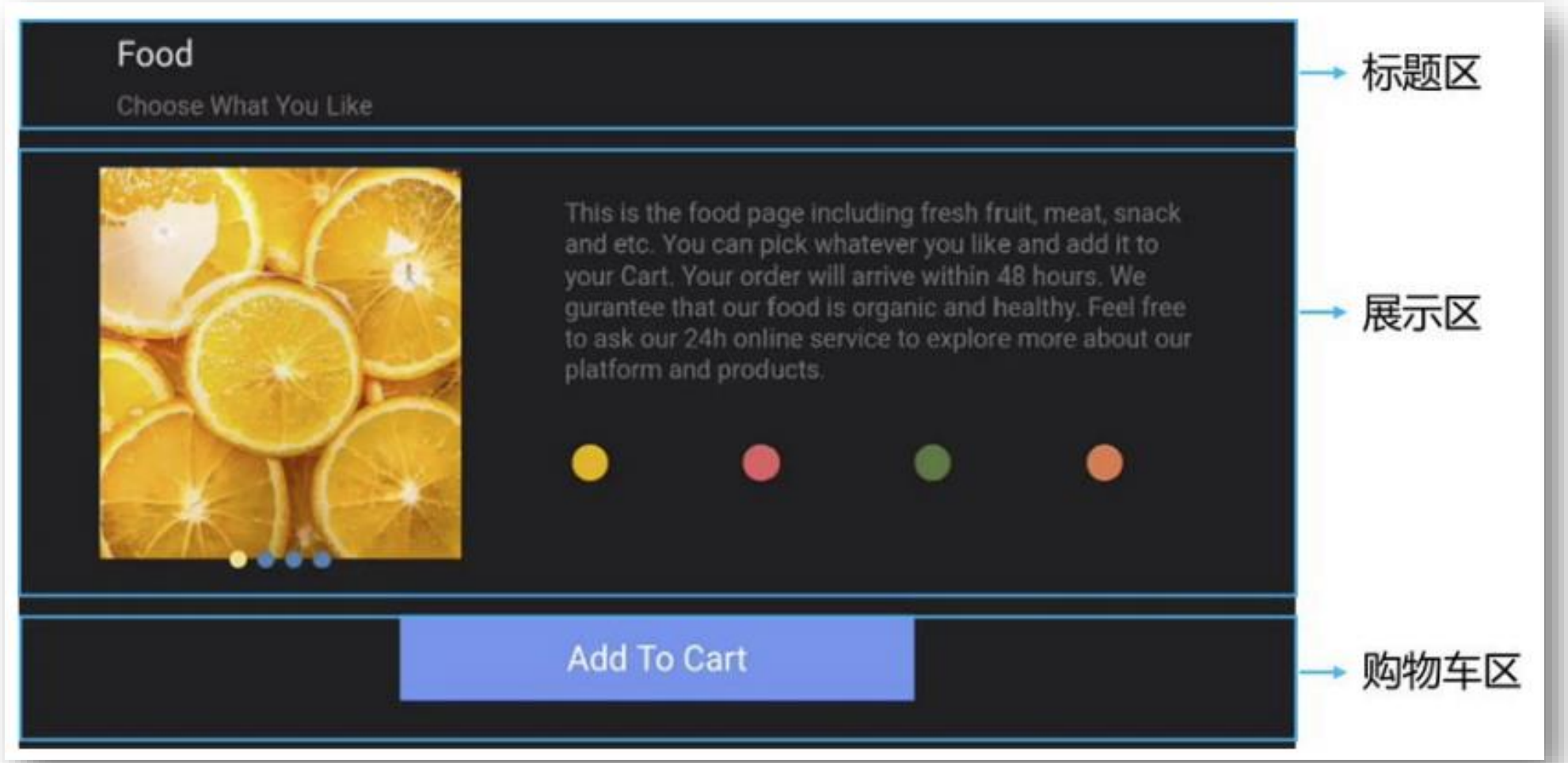
index.hml

```
<div class="container">
    <text class="title">
        {{ $t('strings.hello') }} {{title}}
    </text>
</div>
```

index.js

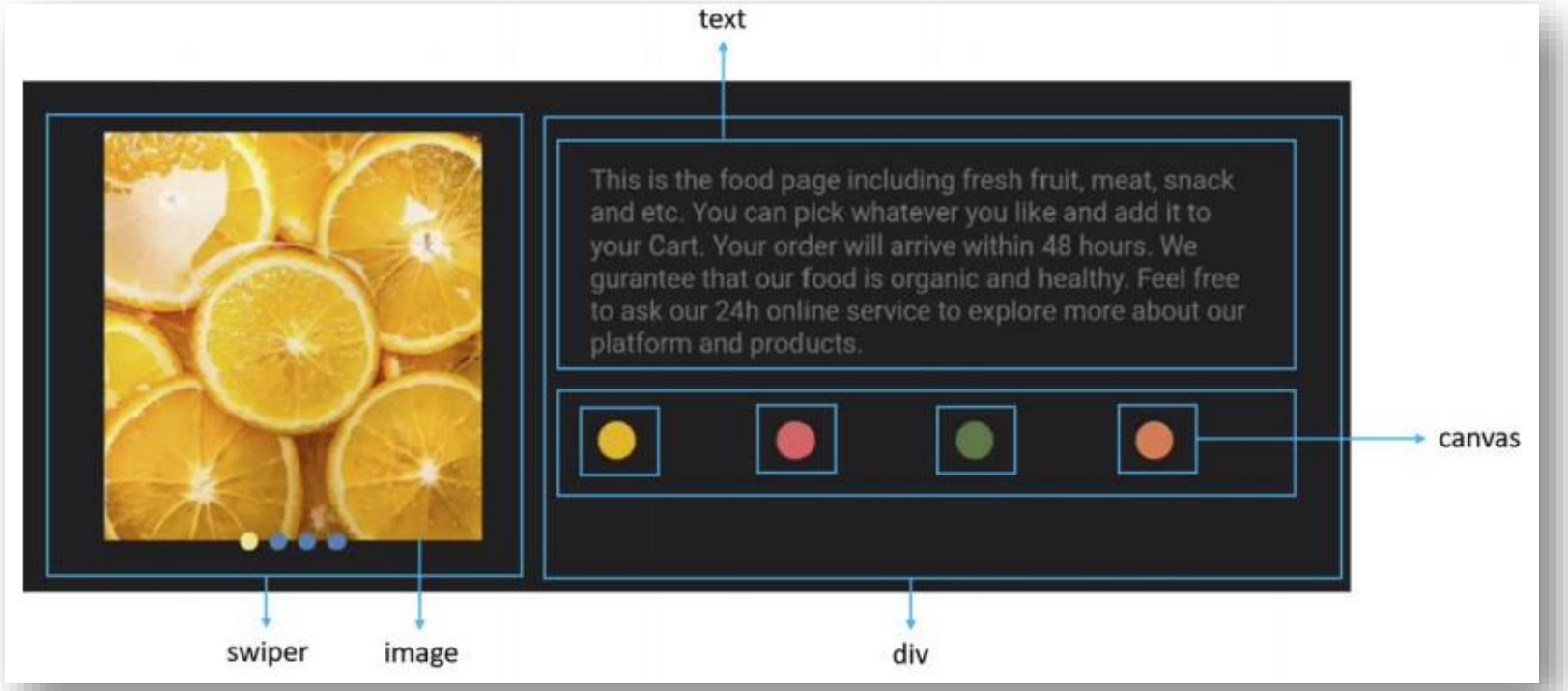
```
export default {
    data: {
        title: ""
    },
    onInit() {
        this.title = this.$t('strings.world');
    }
}
```

# JS FA示例


Food
Choose What You Like
This is the food page including fresh fruit, meat, snack and etc. You can pick whatever you like and add it to your Cart. Your order will arrive within 48 hours. We gurantee that our food is organic and healthy. Feel free to ask our 24h online service to explore more about our platform and products.
Add To Cart
标题区
展示区
购物车区

# JS FA应用示例-构建页面布局

- 在*index.hml*文件构建页面布局，将页面分解为不同部分，用容器组件承载。根据应用效果图，页面分成三个部分：标题区、展示区和购物车区。
- 标题区由两个按列排列的*text*组件实现；购物车区由一个*text*组件构成
- 展示区由按行排列的*swiper*组件和*div*组件组成
  - 第一部分：容器组件*swiper*，包含四个*image*组件
  - 第二部分：容器组件*div*，包含一个*text*组件和四个画布组件*canvas*绘制的圆形



# JS FA应用示例-构建页面布局

```
<swiper id="swiperImage" class="swiper-style">
    <image src="{{$item}}" class="image-mode" focusable="true" for="{{imageList}}"></image>
</swiper>
<div class="container">
    <div class="description-first-paragraph">
        <text class="description">{{descriptionFirstParagraph}}</text>
    </div>
    <div class="color-column">
        <canvas id="{{$item.id}}" onfocus="swipeToIndex({{$item.index}})" class="color-item" focusable="true"
                for="{{canvasList}}"></canvas>
    </div>
</div>
```

```
<div class="cart">
    <text class="{{cartStyle}}" onclick="addCart" onfocus="getFocus" onblur="lostFocus" focusable="true">
        {{cartText}}</text>
</div>
```

# JS FA应用示例-构建页面样式

- 页面样式在*index.css*文件中设定，需要设定的样式主要有*flex-direction*（主轴方向），*padding*（内 边距），*font-size*（字体大小）等
- 构建页面样式采用*css* 伪类写法，当焦点移动到*canvas*组件上时，背景颜色变成白色，也可以在*js*中通过*focus*和*blur*事件动态修改*css*样式来实现同样的效果

```
.container {
    flex-direction: column;
}
.title-section {
    flex-direction: row;
    height: 60px;
    margin-bottom: 5px;
    margin-top: 10px;
}
.title {
    align-items: flex-start;
    flex-direction: column;
    padding-left: 60px;
    padding-right: 160px;
}
.name {
    font-size: 20px;
}
```

```
.swiper-style {
    height: 250px;
    width: 350px;
    indicator-color: #4682b4;
    indicator-selected-color: #f0e68c;
    indicator-size: 10px;
    margin-left: 50px;
}
.image-mode {
    object-fit: contain;
}
.color-column {
    flex-direction: row;
    align-content: center;
    margin-top: 20px;
}
```

```
.cart {
    justify-content: center;
    margin-top: 20px;
}
.cart-text {
    font-size: 20px;
    text-align: center;
    width: 300px;
    height: 50px;
    background-color: #6495ed;
    color: white;
}
.cart-text-focus {
    font-size: 20px;
    text-align: center;
    width: 300px;
    height: 50px;
    background-color: #4169e1;
    color: white;
}
```

# JS FA应用示例-构建页面逻辑

- ***index.js***文件用于构建页面逻辑，主要实现两个逻辑功能：
  （1）当焦点移动到不同颜色的圆形，***swiper***滑动到不同图片
  （2）当焦点移动到购物车区时，“***Add To Cart***”背景颜色变化，点击后文字变化“***Cart + 1***”

```
onShow() {
    this.canvasList.forEach(element => {
        this.drawCycle(element.id, element.color);
    });
},
swipeToIndex(index) {
    this.$element('swiperImage').swipeTo({
        index: index
    });
},
drawCycle(id, color) {
    var greenCycle = this.$element(id);
    var ctx = greenCycle.getContext("2d");
    ctx.strokeStyle = color;
    ctx.fillStyle = color;
    ctx.beginPath();
    ctx.arc(15, 25, 10, 0, 2 * 3.14);
    ctx.closePath();
    ctx.stroke();
    ctx.fill();
},
```

```
addCart() {
    if (this.isCartEmpty) {
        this.cartText = 'Cart + 1';
        this.cartStyle = 'add-cart-text';
        this.isCartEmpty = false;
    }
},
getFocus() {
    if (this.isCartEmpty) {
        this.cartStyle = 'cart-text-focus';
    }
},
lostFocus() {
    if (this.isCartEmpty) {
        this.cartStyle = 'cart-text';
    }
},
```

- 组件是构建页面的核心，每个组件通过对数据和方法的封装，实现独立可视、可交互功能单元
- 组件之间相互独立，可以重复使用，也可以通过组件间合理搭配定义满足业务需求的新组件

| 组件类型 | 主要组件 |
| --- | --- |
| 基础组件 | *text*、*image*、*progress*、*rating*、*span*、*marquee*、*image_animator*、*divider*、*menu*、*chart* |
| 容器组件 | *div*、*list*、*list-item*、*stack*、*swiper*、*tabs*、*tab-bar*、*tab-content*、*popup*、*list-item-group*、*refresh*、*dialog* |
| 媒体组件 | *video* |
| 画布组件 | *canvas* |

# 添加外部容器

- 将页面基本元素组装在一起需要使用容器组件。页面布局常用三种容器组件：*div*、*list*和*tabs*，页面结构相对简单时，可以直接用*div*作为容器

- *list*组件：当页面结构较为复杂时，如果使用 *div* 循环渲染，容易出现卡顿，推荐使用*list*组件代替*div*组件实现长列表布局，从而实现更加流畅的列表滚动体验，*list*组件仅支持*list-item*作为子组件

- *tabs*组件：当页面经常需要动态加载时，推荐使用*tabs*组件，*tabs*组件支持*change*事件，在页签切换后触发。*tabs*组件仅支持一个*tab-bar*和一个*tab-content*

```
1.  <!-- xxx.hml -->
2.  <list class="list">
3.    <list-item type="listItem" for="{{textList}}">
4.      <text class="desc-text">{{$item.value}}</text>
5.    </list-item>
6.  </list>
```

```
1.  <!-- xxx.hml -->
2.  <tabs>
3.    <tab-bar class="tab-bar">
4.      <text style="color: #000000">tab-bar</text>
5.    </tab-bar>
6.    <tab-content>
7.      <image src="{{tabImage}}"></image>
8.    </tab-content>
9.  </tabs>
```

- UI事件包括手势事件和按键事件。手势事件主要用于触摸屏设备，按键事件主要用于智慧屏设备

## 1.手势事件

- 手势表示由单个或多个事件识别的语义动作（例如：点击、拖动和长按），一个完整的手势也可能由多个事件组成，对应手势的生命周期

### （1）触摸事件

- ***touchstart***：触摸动作开始
- ***touchmove***：触摸后移动
- ***touchcancel***：触摸动作被打断，如来电提醒
- ***touchend***：触摸动作结束

### （2）点击事件

- ***click***：用户快速轻敲屏幕

### （3）长按事件

- ***longpress***：相同位置长时间保持与屏幕接触

## 2.按键事件

- 按键事件是智慧屏上特有的事件，操作遥控器按键时触发

# JS FA如何调用PA

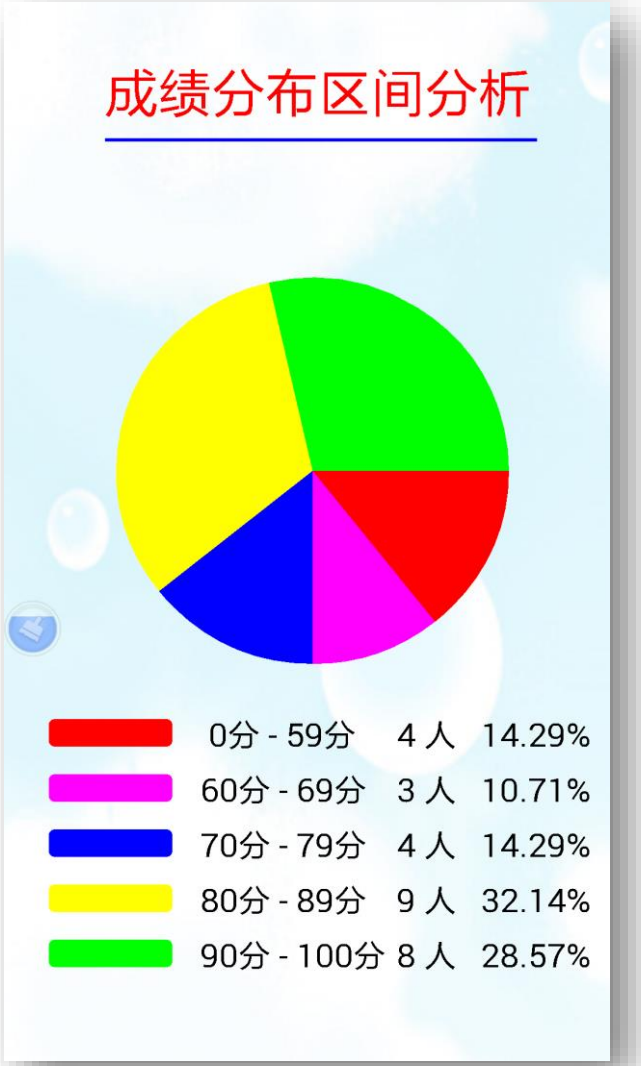
- ***JS UI*** 框架提供了 ***JS FA***（***Feature Ability***）调用 ***Java PA***（***Particle Ability***）的机制，提供一种通道传递方法调用、数据返回以及订阅事件上报，包括***Ability***和***Internal Ability***两种调用方式

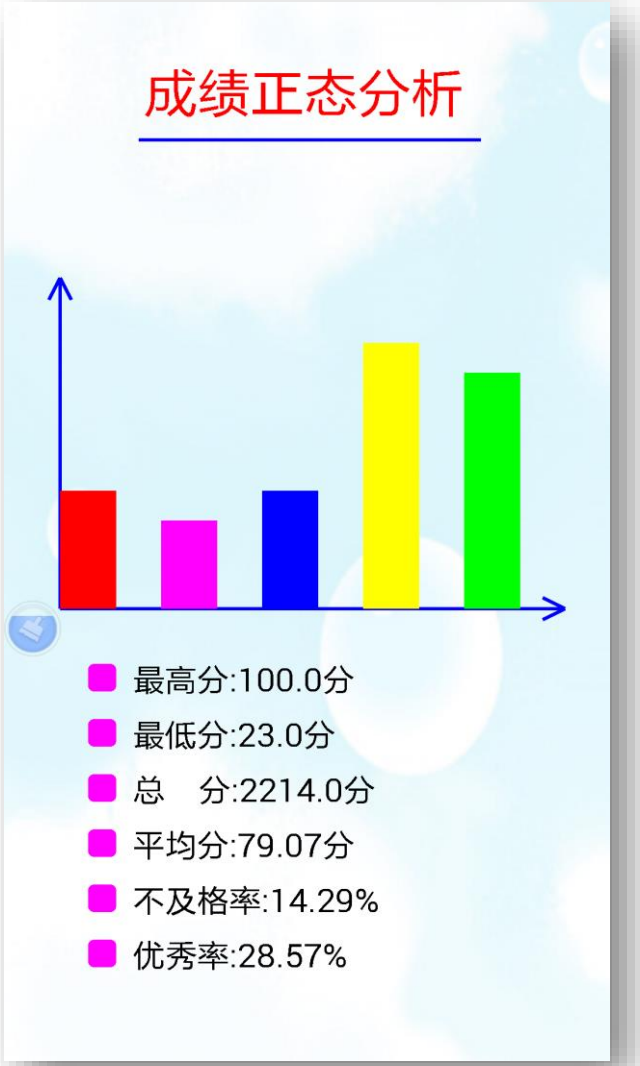
- ***Ability***：拥有独立的***Ability***生命周期，***FA***使用远端进程通信请求***PA***服务，适用于基本服务供多***FA***调用或者服务在后台独立运行的场景

- ***Internal Ability***：与***FA***共进程，采用内部函数调用的方式和***FA***进行通信，适用于对服务响应时延要求较高的场景。 该方式下***PA***不支持其他***FA***访问调用

- ***JS***端与***Java***端通过***bundleName***和***abilityName***来进行关联。系统收到***JS***调用请求后， 根据***JS***接口参数选择对应处理方式

# 前面的UI方法够用吗?




1. 这些效果怎么做?

2. 如果还需要动画呢?

# 主要内容

- ***Harmony JS UI*框架**
- 绘制图形的画布
- 图形绘制
- 图像处理
- 动画基础
- ***OpenGL ES*基础**

# Android 游戏和游戏中的画布

- 游戏是移动互联网应用中最大众化的应用之一，分为2D和3D游戏，2D游戏主要通过画布（***Canvas***）来绘制各种游戏元素，3D游戏通常会借助***OpenGL ES***及更高级的游戏引擎进行渲染，效果更逼真，可玩性更强

- 通过画布（***Canvas***）可以绘制基本图形元素，例如像素点、直线、圆、矩形等，*SDK*提供可操作画布(***Canvas***)的类

# 在View上实现动画效果

- 绘图须首先获得*Canvas*对象，然后通过*Canvas*对象提供的绘图方法绘制图形，通过刷新可更新并重绘画布
- 在*View*上实现动画效果的步骤如下
  - 编写一个*View*类子类
  - 覆盖*View.onDraw()*方法
  - 用多线程处理动画效果
- 注意：线程中不能直接访问*UI*控件，也不能调用任何影响*UI*控件变化的方法

# 在View上实现动画示例

```
public class GameView extends View implements OnTouchListener{
  public float x;   public float y;
  public GameView(Context context){
     super(context); setOnTouchListener(this);}
```

```
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    Paint paint = new Paint();
    paint.setColor(Color.WHITE);
    canvas.drawCircle(x, y, 20, paint);}
```

每次刷新时调用

```
public boolean onTouch(View view, MotionEvent event){
   AnimThread animThread = new AnimThread(this, event.getX(), event.getY());
   Thread thread = new Thread(animThread);
   thread.start();
   return false; }
```

每次触屏时调用

```
private Handler handler = new Handler(){
   public void handleMessage(Message msg) {
     ((View) msg.obj).invalidate();
      super.handleMessage(msg);}};
```

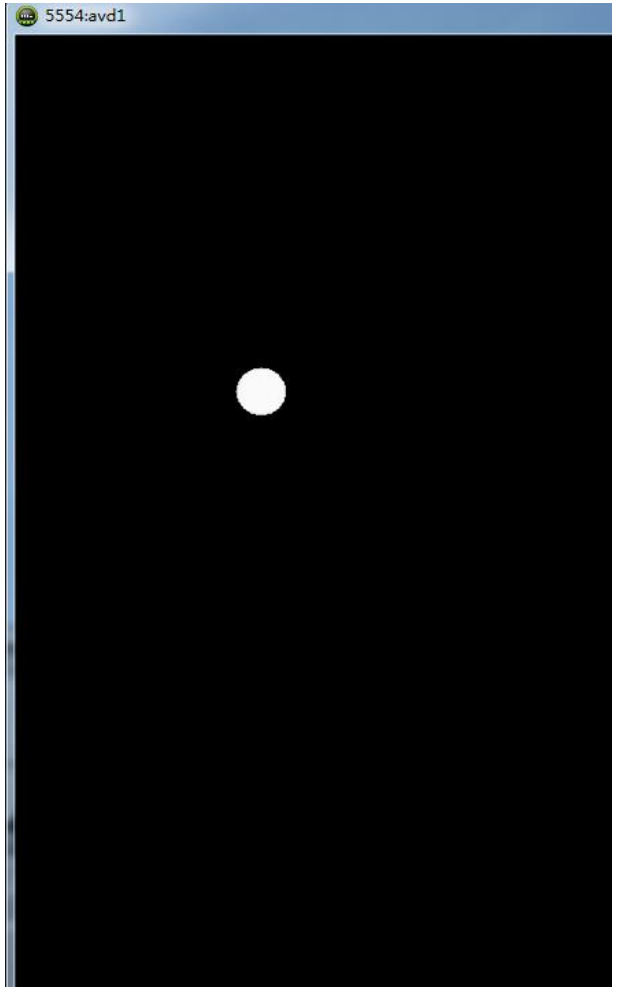
多线程回调刷新调用

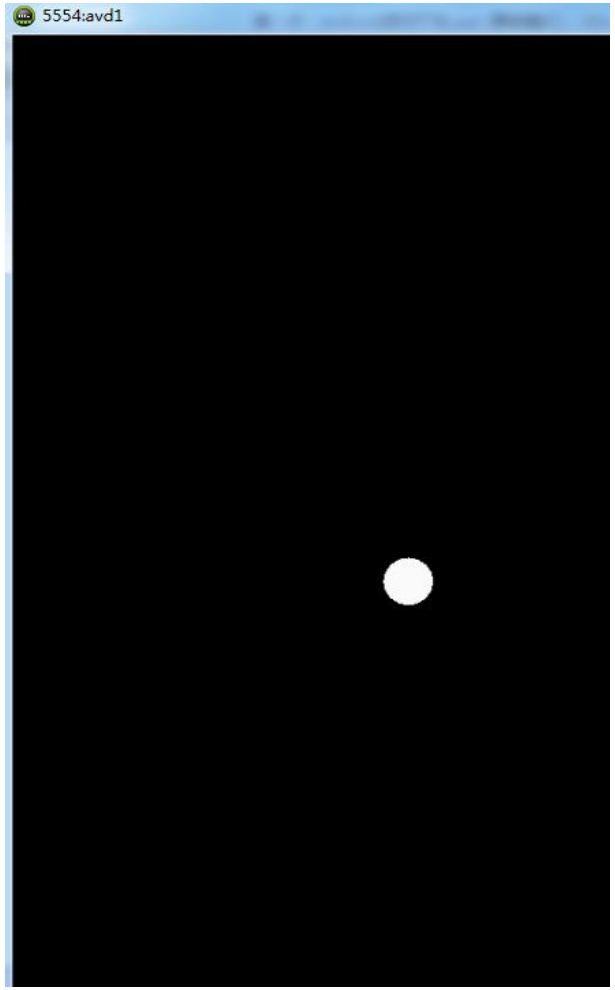
# 在View上实现动画示例

```
class AnimThread implements Runnable{
   private float newX, newY;
   private View view;
   public AnimThread(View view, float newX, float newY){
      this.newX = newX;  this.newY = newY; this.view = view;}
   public void run(){
     float scale = Math.abs(newY - y) / Math.abs(newX - x);
    while (newX != x && newY != y) {
      //计算动画新坐标
      try{
         Message msg = new Message();
         msg.obj = view;
         handler.sendMessage(msg);
         Thread.sleep(50);
      }catch (Exception e){}
   }
}
```

多线程回调刷新

# 在View上实现动画示例



# 用高层SDK实现动画效果

- 实现游戏动画还可使用更高层的SDK，*Android*中*SurfaceView*比*View*更有优势：
  - 可直接获得*Canvas*对象
  - 支持双缓冲技术，绘制图形效率更高
  - 可以在非*UI*线程中直接绘制图形，而*View*必须要使用*Handler*对象发送消息，通知*UI*线程绘制图形

# 在SurfaceView上实现动画示例

```
class AnimThread implements Runnable{
    private float newX, newY;
    public AnimThread(float newX, float newY){
       this.newX = newX;  this.newY = newY; }
   public void run(){ //计算绘制动画的坐标
     try{drawCircle();
         Thread.sleep(50);}
     catch (Exception e){}  }
   public void drawCircle(){
      if (surfaceHolder == null)return;
     Canvas canvas = surfaceHolder.lockCanvas();    //获得Canvas对象
     canvas.drawColor(Color.BLACK);                 //清空屏幕
     if (canvas == null)return;
     Paint paint = new Paint();
     paint.setColor(Color.WHITE);                   //绘制图形
     canvas.drawCircle(x, y, 20, paint);
     surfaceHolder.unlockCanvasAndPost(canvas);  //释放Canvas对象，并将缓冲区绘制的
                                                    图形一次性绘制到画布上
}
```

非UI线程直接调用绘图

# 主要内容

- ***Harmony OS JS UI*框架**
- 绘制图形的画布
- 图形绘制
- 图像处理
- 动画基础
- OpenGL ES基础

- 通过***Canvas***对象可绘制基本图形、文本、位图，无论使用***View***还是使用***SurfaceView***，***Canvas***的使用方法均相同

（1）绘制像素点：像素点是一切图形元素的基础，使用***Canvas***.*drawPoint*()可以在指定的坐标绘制像素点，或指定一组坐标绘制多个像素点

（2）绘制直线：使用***Canvas***.*drawLine*()可绘制一条或者多条直线

（3）绘制圆和弧：使用***Canvas***.*drawCircle*()可绘制圆，使用***Canvas***.*drawArc*()可绘制弧

（4）绘制文本：使用***Canvas***.*drawText*()或***Canvas***.*drawPosText*()可绘制文本

# 图形绘制示例


11:20
画图
圆
形
Android（宽度：55.0）
Android（宽度：66.0）
Android（宽度：74.0）
Android（宽度：87.0）
Android（宽度：98.0）


11:20
画图
圆
形
Android（宽度：98.0）
Android（宽度：87.0）
Android（宽度：74.0）
Android（宽度：66.0）
Android（宽度：55.0）

# 主要内容

- ***Harmony OS JS UI*框架**
- 绘制图形的画布
- 图形绘制
- 图像处理
- 动画基础
- ***OpenGL ES*基础**

# 设置图像的属性

- 图像透明度：图像的透明度分为255个等级，用整型值表示，值越小表示越透明，0表示全透明，完全透明的位图将从*View*上消失

```
Paint.setAlpha(int alpha)
```

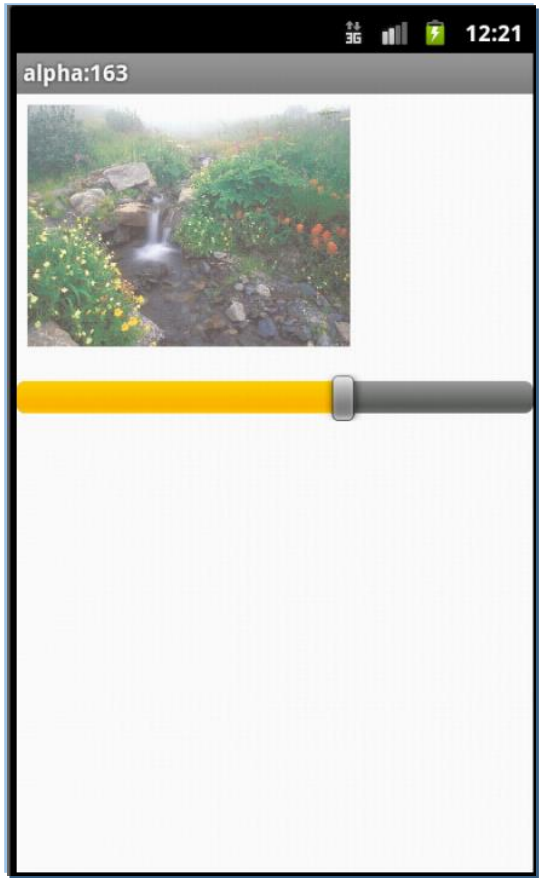
- 图像旋转：用*Matrix*.*setRotate()*可对图像进行任意角度的旋转

```
Matrix.setRotate(float degrees)
Matrix.setRotate(float degrees, float px, float py)
```

*degrees*表示旋转角度，为正表示顺时针旋转，否则为逆时针旋转，默认为绕图像中心点旋转；*px,py*表示图像旋转的轴心坐标

```
bitmap=Bitmap.createBitmap(bitmap,0,0,bitmap.getWidth(),
                                        bitmap.getHeight(),matrix,true);
canvas.setMatrix(matrix);
canvas.drawBitmap(bitmap,0,0,null);
```

# 绘制位图示例


# 绘制路径

- ***Canvas***提供了绘制路径的功能，通过***Canvas***.*drawPath*方法，可以画出封闭路径和开放路径，并可以在路径上实现特殊效果

***public void drawPath(Path path, Paint paint);***

*path*用于绘制路径的轨迹，对于开放路径，需要绘制组成路径的多条线段，对于封闭路径，需要绘制封闭路径的形状，*paint*参数用于指定特效、颜色等路径属性

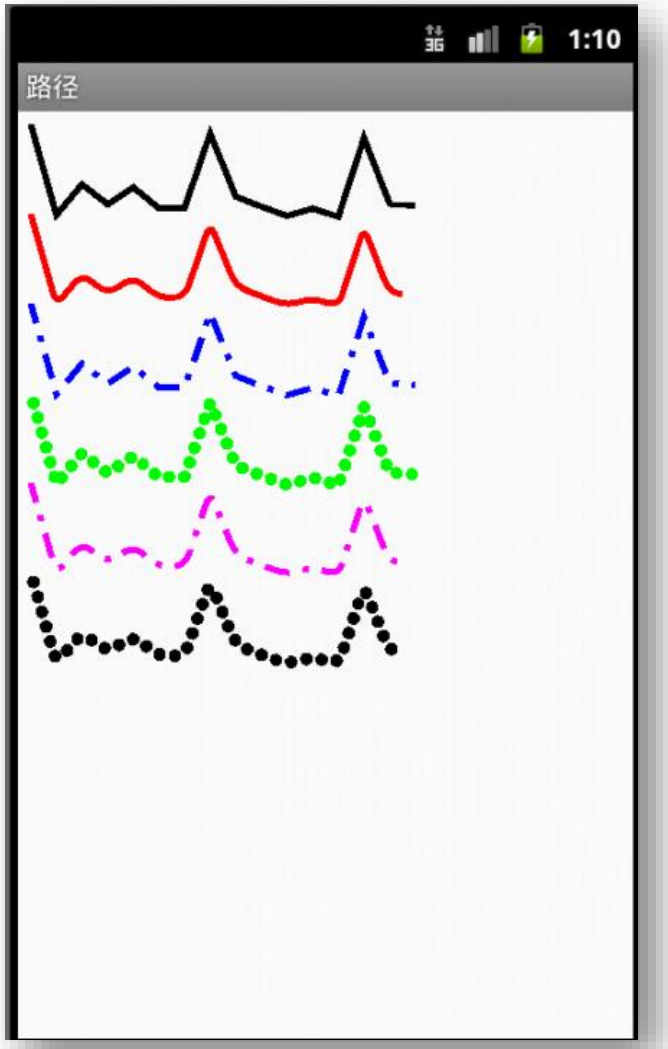
```
private Path makeFollowPath(){
    Path p = new Path();
    p.moveTo(0, 0);
   for (int i = 1; i <= 15; i++){
     p.lineTo(i * 20, (float) Math.random() * 75);
   }
  return p;
}
```

# 路径特效

- 路径特效用*PathEffect*对象表示，该对象具有很多子类，分别表示不同的特效，可通过如下方法设置

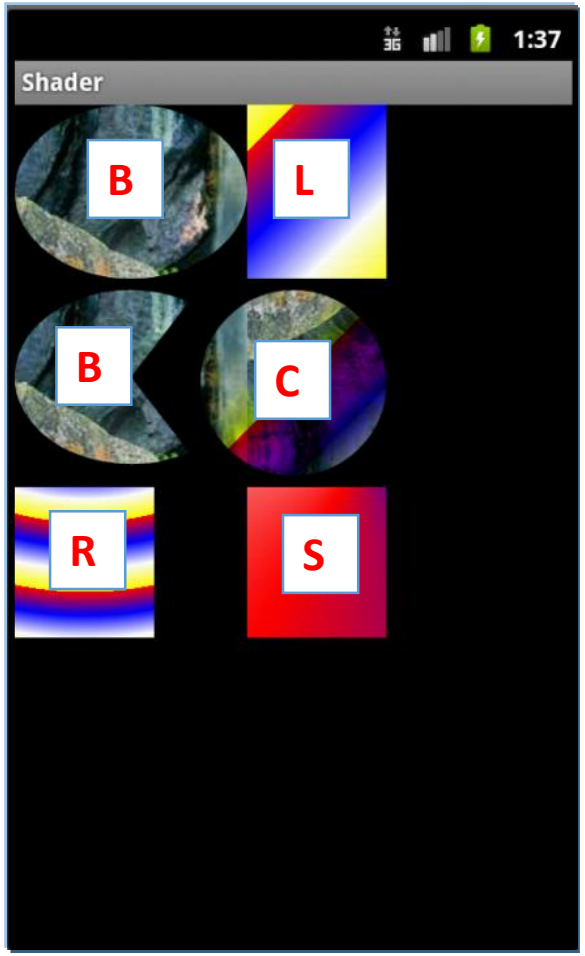
***private void makeEffects(PathEffect[] e, float phase)***

- *e[0]=null* 表示无特效
- *e[1]= new CornerPathEffect()* 表示圆角效果
- *e[2] = new DashPathEffect()* 表示虚线路径
- *e[3]=PathDashPathEffect()* 表示虚线路径图形
- *e[4] = new ComposePathEffect(e[2],e[1])* 表示组合特效



# Shader渲染效果

- 提供图像渲染功能
  - *BitmapShader*：将图像按照椭圆或弧方式绘制,从图像左上角截取图形绘制，绘制模式为*TileMode.REPEAT*, *TileMode.MIRROR*
  - *LinearGradient*：线性渐变效果
  - *RadialGradient*：径向渐变效果
  - *SweepGradient*：角度渐变效果（光照效果）
  - *ComposerShader*：将两种渲染效果组合在一起



# 主要内容

- ***Harmony OS JS UI*框架**
- 绘制图形的画布
- 图形绘制
- 图像处理
- 动画基础
- ***OpenGL ES*基础**

# 帧动画

- 帧动画是由若干幅图像组成的动画，这些图形以一定的时间间隔进行切换，每秒不少于可以看做是连续动画25帧的动画播放速度
- 帧动画需要在一个**动画文件**中指定构成动画的静态图像和每一个静态图像的停留时间(单位:*ms*)，动画文件为*xml*格式，放在*res/anim*目录中

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android" android:oneshot="false">
  <item android:drawable="@drawable/image1" android:duration="200" />
  <item android:drawable="@drawable/image2" android:duration="100" />
  <item android:drawable="@drawable/image3" android:duration="200" />
  <item android:drawable="@drawable/image4" android:duration="150" />
  <item android:drawable="@drawable/image5" android:duration="100" />
  <item android:drawable="@drawable/image6" android:duration="200" />
</animation-list>
```

- 装载动画文件并创建*AnimationDrawable*对象

```
animationDrawable=(AnimationDrawable)getResources().getDrawable(R.anim.test);
```

- 将*AnimationDrawable*对象作为*ImageView*控件的背景

```
ImageView imageview = (ImageView) findViewById(R.id.imageview);
imageview.setBackgroundDrawable(animationDrawable);
```

- 通过*AnimationDrawable*对象控制帧动画
  - *start*：开始播放帧动画
  - *stop*：停止播放帧动画
  - *setOnShot*：设置是否循环播放帧动画
  - *addFrame*：添加新的帧（单幅图像或动画）到动画中
  - *isOneShot*：判断帧动画是否循环
  - *isRunning*：判断帧动画是否正在播放
  - *getNumOfFrames*：返回动画的帧数
  - *getFrame*：根据索引获得指定帧对象
  - *getDuration*：获得指定帧的停留事件

# 补间（Tween）动画

- 动画中的图像变化有规律，可采用自动生成中间图像的方式生成动画，如移动、旋转、缩放或一定的数学算法，补间动画只需要指定动画的第一帧和最后一帧图像

- 补间动画的主要表现形式有
  - 移动补间动画
  - 缩放补间动画
  - 旋转补间动画
  - 透明度补间动画

# 移动补间动画

```
//动画文件translate_tween.xml
<translate xmlns:android="http://schemas.android.com/apk/res/android"
   android:interpolator="@android:anim/accelerate_interpolator"
   android:fromXDelta="0" android:toXDelta="320" android:fromYDelta="0"
   android:toYDelta="0" android:duration="2000" />
```

- android:interpolator表示动画渲染器，可设置为动画加速器(accelerate_interpolator)，动画减速器(decelerate_interpolator)和动画加速减速器(accelerate_decelerate_interpolator)

```
Animation animation = AnimationUtils.loadAnimation(this, R.anim.translate_tween);
animation.setRepeatCount(Animation.INFINITE);
EditText editText = (EditText)findViewById(R.id.edittext);
editText.startAnimation(animation);
```

# 缩放补间动画

```
//动画文件to_large.xml
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:fromXScale="0.2" android:toXScale="1.0" android:fromYScale="0.2"
    android:toYScale="1.0" android:pivotX="50%" android:pivotY="50%"   //缩放支点
    android:duration="500" />

imageView = (ImageView) findViewById(R.id.imageview);
toLargeAnimation = AnimationUtils.loadAnimation(this, R.anim.to_large);
toSmallAnimation = AnimationUtils.loadAnimation(this, R.anim.to_small);
toLargeAnimation.setAnimationListener(this);
toSmallAnimation.setAnimationListener(this);
imageView.startAnimation(toSmallAnimation);

if (animation.hashCode() == toLargeAnimation.hashCode())
    imageView.startAnimation(toSmallAnimation);
else
    imageView.startAnimation(toLargeAnimation);
```

# 旋转补间动画

```
//动画文件rotate_tween.xml
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
  android:interpolator="@android:anim/linear_interpolator" android:fromDegrees="0"
  android:toDegrees="360" android:pivotX="50%" android:pivotY="50%"
  android:duration="10000" android:repeatMode="restart" android:repeatCount="infinite"/>
```

```
ImageView imageView = (ImageView)findViewById(R.id.imageview);
Animation animation = AnimationUtils.loadAnimation(this,R.anim.rotate_tween);
imageView.startAnimation(animation);
```

- android:repeatMode可设置为restart或者reverse
- android:repeatCount可设置旋转次数，通常设置为某一整数值，若设置为infinite或-1，表示动画永不停止

# 透明度补间动画

```
//动画文件alpha_tween.xml
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
  android:interpolator="@android:anim/accelerate_interpolator"
      android:fromAlpha="1.0" android:toAlpha="0.1" android:duration="5000" />
```

```
ImageView imageView = (ImageView)findViewById(R.id.imageview);
Animation animation = AnimationUtils.loadAnimation(this,R.anim.alpha_tween);
imageView.startAnimation(animation);
```

- ***Harmony Java UI*** 框架提供了数值动画(***AnimatorValue***)和属性动画(***AnimatorProperty***)，并提供将多个动画同时操作的动画集合(***AnimatorGroup***)
- 数值动画

***AnimatorValue*** 数值从 0 到 1 变化，本身与 ***Component*** 无关。可以设置 0 到 1 变化过程的属性，例如时长、变化曲线、重复次数等，并通过值的变化改变组件的属性，实现组件的动画效果

- 属性动画

***Java UI*** 框架以为 ***Component*** 设置某个属性或多个属性的动画

```
AnimatorValue animator = new AnimatorValue();
animator.setDuration(2000);
animator.setDelay(1000);
animator.setLoopedCount(2);
animator.setCurveType(Animator.CurveType.BOUNCE);
……
//添加回调函数
button.setContentPosition((int) (800 * value),
                          button.getContentPositionY());
……
animator.start();    //启动动画
```


```
AnimatorProperty animator = button.createAnimatorProperty();
animator.moveFromX(50).moveToX(1000).rotate(180)
        .setDuration(1000).setDelay(500).setLoopCount(5);
animator.start();
```

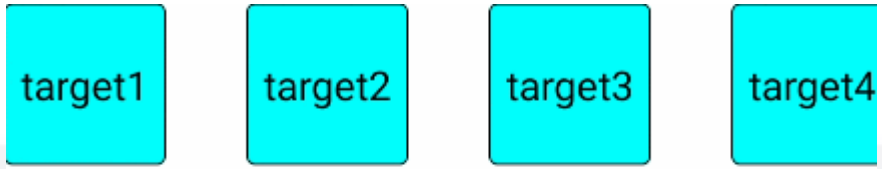
- 可以把多个动画对象组合并添加到*AnimatorGroup*。*AnimatorGroup*提供了两个方法：*runSerially()*和*runParallel()*，分别表示动画按顺序开始和动画同时开始。

1. 声明 AnimatorGroup。

```
AnimatorGroup animatorGroup = new AnimatorGroup();
```

2. 添加要按顺序或同时开始的动画。

```
// 4 个动画按顺序播放
animatorGroup.runSerially(am1, am2, am3, am4);
// 4 个动画同时播放
animatorGroup.runParallel(am1, am2, am3, am4);
```




3. 启动动画或对动画做其他操作。

```
animatorGroup.start();
```

# 静态动画

- 动画分为静态动画和连续动画

- 静态动画的核心是***transform***样式，可以实现三种变换类型，一次样式设置只能实现一种类型变换
  - (1) ***translate***：沿水平或垂直方向将指定组件移动所需距离
  - (2) ***scale***：横向或纵向将指定组件缩小或放大到所需比例
  - (3) ***rotate***：将指定组件沿横轴或纵轴或中心点旋转指定的角度

```
.translate {
    height: 300px;
    width: 400px;
    font-size: 100px;
    background-color: #008000;
    transform: translate(300px);
}
```

```
.rotate {
    height: 300px;
    width: 400px;
    font-size: 100px;
    background-color: #008000;
    transform-origin: 200px 100px;
    transform: rotateX(45deg);
}
```

```
.scale {
    height: 300px;
    width: 400px;
    font-size: 100px;
    background-color: #008000;
    transform: scaleX(1.5);
}
```

# 连续动画

- 连续动画的核心是*animation*样式，定义了动画的开始状态、结束状态以及时间和速度的变化曲线

（1）*animation-name*：设置动画执行后应用到组件上的背景颜色、透明度、宽高和变换类型

（2）*animation-delay*和*animation-duration*：分别设置动画执行后元素延迟和持续的时间

（3）*animation-timing-function*：描述动画执行的速度曲线，使动画更加平滑

（4）*animation-iteration-count*：定义动画播放的次数

（5）*animation-fill-mode*：指定动画执行结束后是否恢复初始状态

```
1. <!—xxx.hml-->
2. <div class="item-container" >
3.   <div class="group">
4.     <text class="header">animation-name </text>
5.     <div class="item {{colorParam}}" >
6.       <text class="txt">color </text>
7.     </div>
8.     <div class="item {{opacityParam}}" >
9.       <text class="txt">opacity </text>
10.    </div>
11.    <input class="button" type="button" name="" value="show" onclick="showAnimation” />
12.  </div>
13. </div>
```

定义动画开始状态、结束状态以及时间和速度的变化曲线

设置动画持续时间

```
31.  .color {
32.      animation-name: Color;
33.      animation-duration: 8000ms;
34.   }
35.  .opacity {
36.     animation-name: Opacity;
37.     animation-duration: 8000ms;
38.  }
```

设置动画过渡效果

```
39. @keyframes Color {
40.    from {
41.       background-color: #f76160;
42.    }
43.    to {
44.       background-color: #09ba07;
45.    }
46. }
```

```
47.   @keyframes Opacity {
48.       from {
49.          opacity: 0.9;
50.       }
51.       to {
52.          opacity: 0.1;
53.       }
54.   }
```

# 主要内容

- ***Harmony OS JS UI*框架**
- 绘制图形的画布
- 图形绘制
- 图像处理
- 动画基础
- OpenGL ES基础

# 什么是OpenGL ES?

- 是*OpenGL*的移动版本，是用于编写*3D*图形程序的*API*。*OpenGL ES*广泛用于移动设备，*Android*、*iPhone*的*SDK*都集成了*OpenGL ES API*

- *OpenGL ES*的坐标系是三维的，屏幕中心点是其坐标原点，将手机屏幕朝上正放在桌面上
  - *x*轴是手机屏幕从左到右的方向
  - *y*轴是手机屏幕从下到上的方向
  - *z*轴是从桌面到天空的方向

（1）初始化：建立一个***MyRender***类（***Renderer***类的子类），在*onSurfaceChanged*方法中实现初始化

```
public void onSurfaceChanged(GL10 gl, int width, int height) {
    gl.glViewport(0, 0, width, height);                    //设置OpenGL视窗大小
    gl.glMatrixMode(GL10.GL_PROJECTION);                   //设置投影矩阵
    gl.glFrustumf(-1, 1, -1, 1, 1, 10);                    //设置坐标轴的最远距离
    gl.glMatrixMode(GL10.GL_MODELVIEW);                    //选择模型观察矩阵
}
```

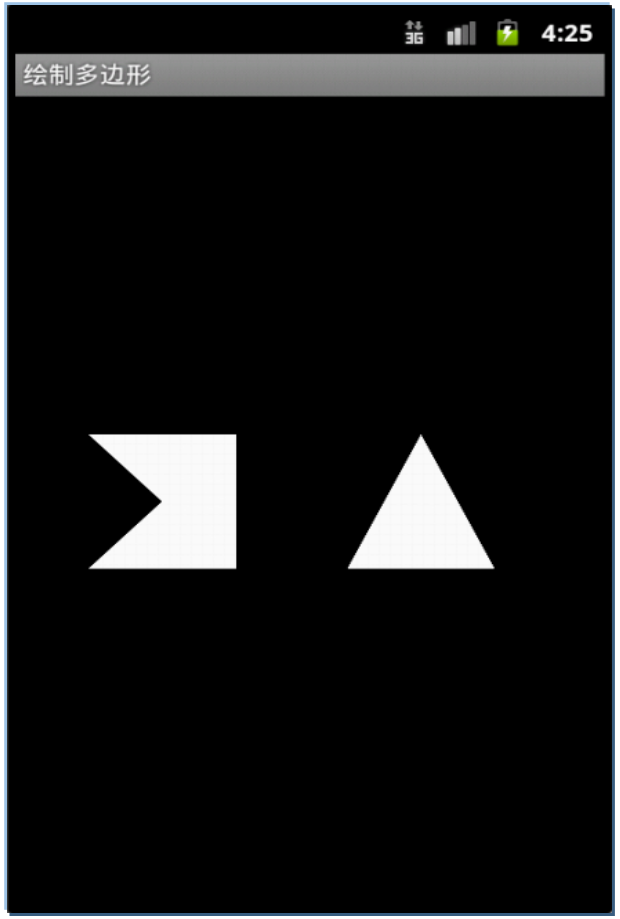
## （2）定义图形并初始化*IntBuffer*

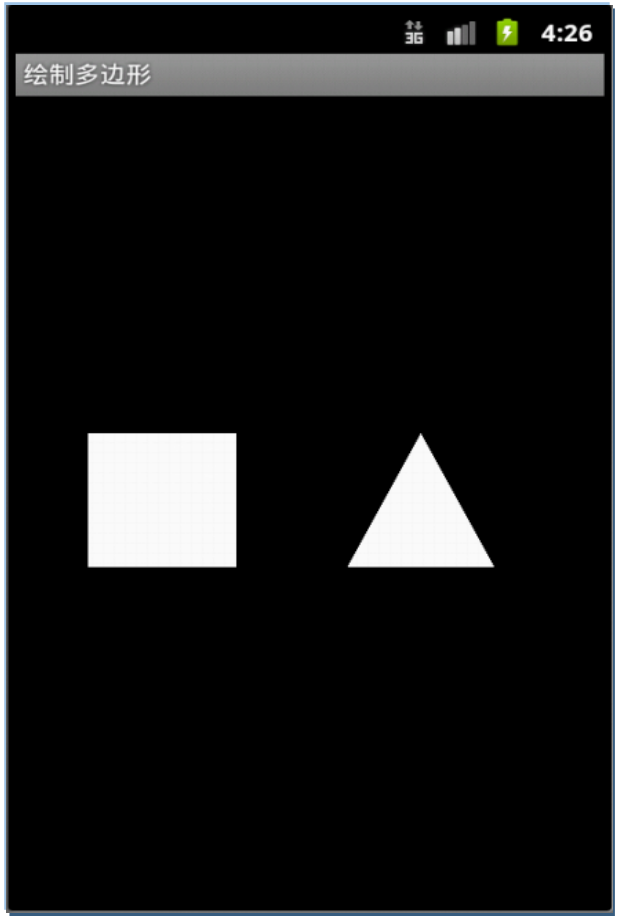
```
private int[] triangleVertices = new int[] { 0, 1, 0, -1, -1, 0, 1, -1, 0 };
private IntBuffer triangleBuffer;

public void onSurfaceCreated(GL10 gl, EGLConfig config){
ByteBuffer byteBuffer = ByteBuffer.allocateDirect(triangleVertices.length * 4);
byteBuffer.order(ByteOrder.nativeOrder());          //返回ByteBuffer的字节序
triangleBuffer = byteBuffer.asIntBuffer();
triangleBuffer.put(triangleVertices);
triangleBuffer.position(0);
}
```

## （3）绘制图形

```
public void onDrawFrame(GL10 gl){

  gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT); //清除屏幕和深度缓存
  gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);                    //允许设置顶点

  gl.glLoadIdentity();                                             //重置当前模型观察矩阵
  gl.glTranslatef(1.5f, 0.0f, -6.0f);
  gl.glVertexPointer(3, GL10.GL_FIXED, 0, triangleBuffer);         //设置三角形顶点坐标
  gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);                        //绘制三角形

  gl.glLoadIdentity();                                             //重置当前的模型观察矩阵
  gl.glTranslatef(-2.0f, 0.0f, -6.0f);
  gl.glVertexPointer(3, GL10.GL_FIXED, 0, quaterBuffer);
  gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 0, 4);                     //绘制正方形
  gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);                   //关闭顶点设置功能
}
```

# 用OpenGL ES绘制图形示例




- gl.glDrawArrays(GL10.***GL_TRIANGLE_FAN***, *0, 4);  //p1p2p3, p1p3p4画三角形*
- gl.glDrawArrays(GL10.***GL_TRIANGLE_STRIP***, *0, 4); //p1p2p3,p2p3p4画三角形*

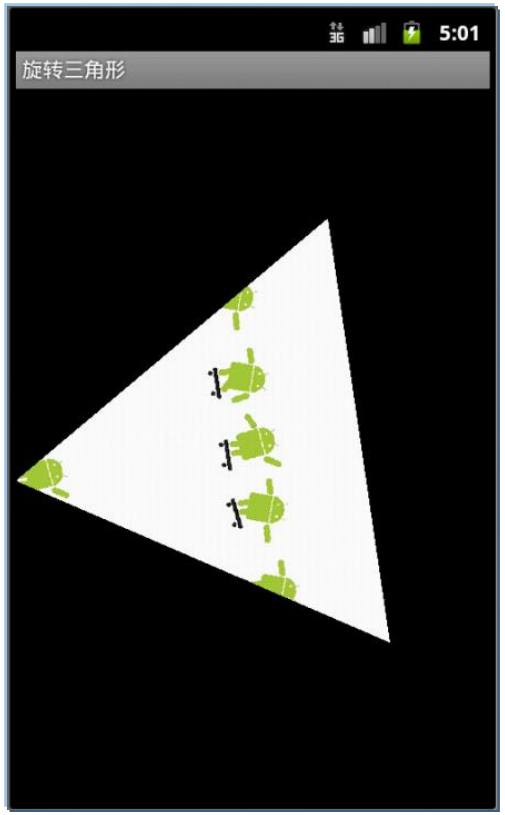
# OpenGL ES的颜色

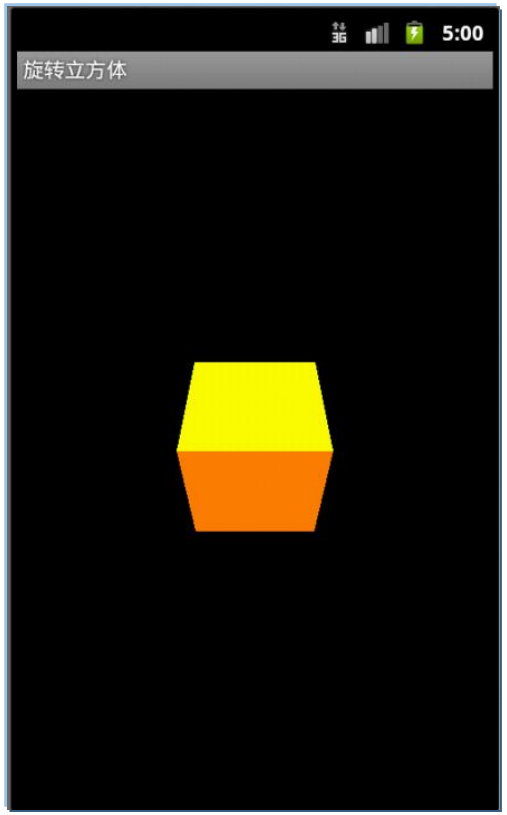
## ◘ *OpenGL ES*的颜色格式均为*RGBA*格式

```
//定义颜色数组
int one = 0x10000;
private IntBuffer colorBuffer;
private int[] colorVertices = new int[]{ 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1 };

//开启颜色渲染功能
gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
gl.glColorPointer(4, GL10.GL_FIXED, 0, colorBuffer);

//关闭颜色渲染功能
gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
gl.glColor4f(1.0f, 0.0f, 0.0f, 0.0f);
```

# OpenGL ES旋转图形

```
long time = SystemClock.uptimeMillis() % 4000L;
float angle = 0.090f * ((int) time);
glRotatef(angle, 0, 0, 1.0f);  //绕Z轴旋转
```



# The End