# 实验十五　ArkUI 动画基础

## 一、实验目的

1. 了解 DevEco Studio 的使用
2. 学习 ArkTS 语言及 ArkUI 动画基础
3. 编写代码
4. 编译运行
5. 在模拟器上运行

## 二、实验原理

1. 鸿蒙开发原理
2. ArkTS，ArkUI 开发原理
3. 鸿蒙应用运行原理

## 三、实验仪器材料

1. 计算机实训室电脑一台
2. DevEco Studio 开发环境及鸿蒙手机模拟器

## 四、实验步骤

参考官方文档：https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-animation-V5

设计原理参考：
https://developer.huawei.com/consumer/cn/doc/design-guides/animation-attributes-0000001797117229

### 1. 打开 DevEco Studio，点击 Create Project 创建工程

设置项目名称为 AnimationDemo。

ArkUI 中提供多种动画接口（属性动画、转场动画等），用于驱动属性值按照设定的动画参数，从起始值逐渐变化到终点值。尽管变化过程中参数值并非绝对的连续，而是具有一定的离散性。但由于人眼会产生视觉暂留，所以最终看到的就是一个“连续“的动画。UI 的一次改变称为一个动画帧，对应一次屏幕刷新。决定动画流畅度的一个重要指标就是帧率 FPS（Frame Per Second），即每秒的动画帧数，帧率越高则动画就会越流畅。


- 属性动画：最基础的动画类型，按照动画参数逐帧驱动属性的变化，产生一帧帧的动画效果。
- 转场动画：为组件在出现和消失时添加过渡动画。为了保证动画一致性，部分接口动画曲线已内置，不支持开发者自定义。
  - 不推荐在应用内使用 UIAbility 组合所有的界面：UIAbility 是一个任务，会在多任务界面独立显示一个卡片，UIAbility 之间的跳转是任务之间的跳转。以应用内查看大图的典型场景为例，不建议应用内调用图库的 UIAbility 去打开图片查看大图，会导致任务的跳转，图库的 UIAbility 也会加入多任务界面中。正确的方式是应用内构建大图组件，通过模态转场去调起大图组件，一个任务内的所有的界面都在一个 UIAbility 内闭环。
  - 导航转场中，应使用 Navigation 组件实现转场动画。过去的 page+router 方式在实现导航转场过程中，因为 page 和 page 之间相互独立，其联动动画效果受限。不仅容易导致页面之间的割裂，并且不支持一次开发多端部署。
- 组件动画：组件提供默认动效（如 List 的滑动动效）便于开发者使用，同时部分组件还支持定制化动效。
- 动画曲线：介绍传统曲线和弹簧曲线的特点和使用方式。动画曲线影响属性值的运动规律，进

而决定界面的动画效果。

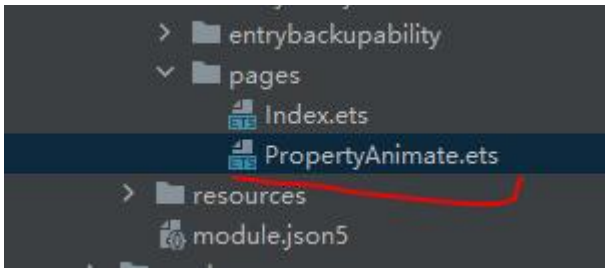
- 动画衔接：介绍如何实现动画与动画之间、手势与动画之间的自然过渡。
- 高阶动画效果：介绍模糊、大阴影和颜色渐变等高阶效果接口的使用方法。

## 2. 属性动画-animateTo()方法

理论讲解部分参考：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-attribute-animation-overview-V5

我们尽量用实例演示。创建一个新的文件：PropertyAnimate.ets



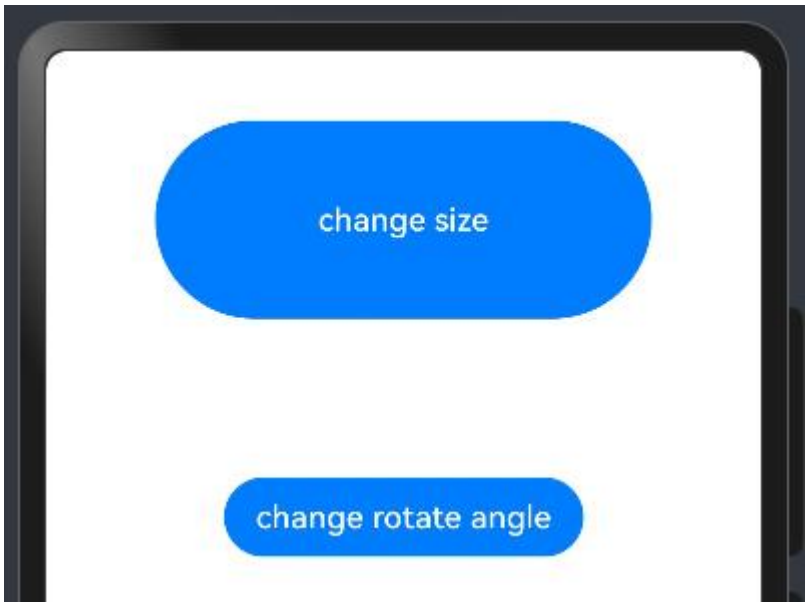

添加基础代码：

```
@Entry
@Component
struct PropertyAnimate {
  @State widthSize: number = 250
  @State heightSize: number = 100
  @State rotateAngle: number = 0

  build() {
    Column() {
      Button('change size')
```

```
            .width(this.widthSize)
            .height(this.heightSize)
            .margin(30)
        Button('change rotate angle')
            .margin(50)
            .rotate({ x: 0, y: 0, z: 1, angle: this.rotateAngle })
      }.width('100%').margin({ top: 5 })
    }
}
```

此时的 Previewer 效果：



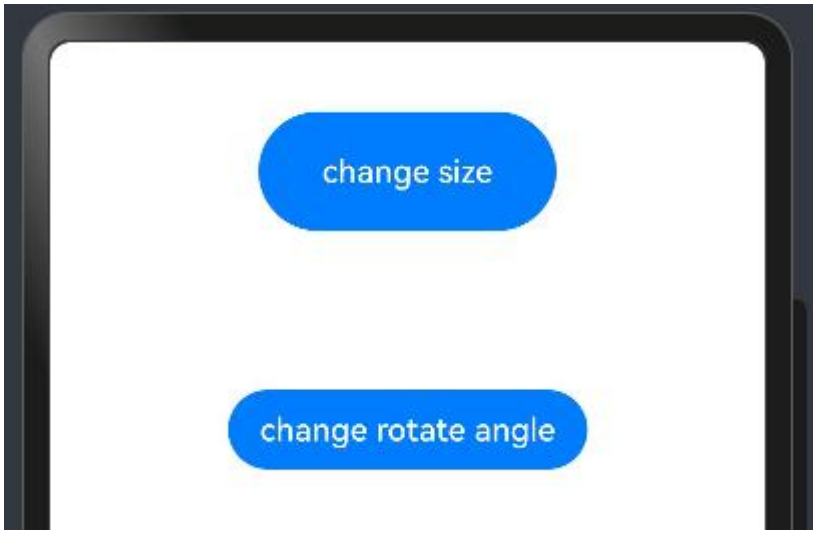

给第一个按钮添加点击事件：

```
            .onClick(() => {
                animateTo({
                    duration: 2000, // 时长 2 秒
```

```
            curve: Curve.EaseOut, // 动画曲线为动画以低速结束
            iterations: 3, // 重复次数，-1 为无限循环
            playMode: PlayMode.Normal, // 播放模式，Normal 表示播放完成后重头开始播放
            onFinish: () => {
              console.info('play end')
            }
          }, () => {
            this.widthSize = 150  // 调整属性目标宽度
            this.heightSize = 60  // 调整属性目标高度
          })
        })
```

请注意看 animateTo 函数里面的参数的含义，后面还跟一个箭头函数，里面加上属性值的目标值的设定。

效果：（重复三次后变为）



我们也可以加一个 flag 去控制再次点击按钮，让其恢复原值。先在 build()前面添加一个参数：

```
  private flag: boolean = true
```

然后将 onClick()事件改为：

```
        .onClick(() => {
          if (this.flag) {
            animateTo({
              duration: 2000,
              curve: Curve.EaseOut,
              iterations: 3,
              playMode: PlayMode.Normal,
              onFinish: () => {
                console.info('play end')
              }
            }, () => {
              this.widthSize = 150
              this.heightSize = 60
            })
          } else {
            animateTo({}, () => {
              this.widthSize = 250
              this.heightSize = 100
            })
          }
```

```
            this.flag = !this.flag
        })
```

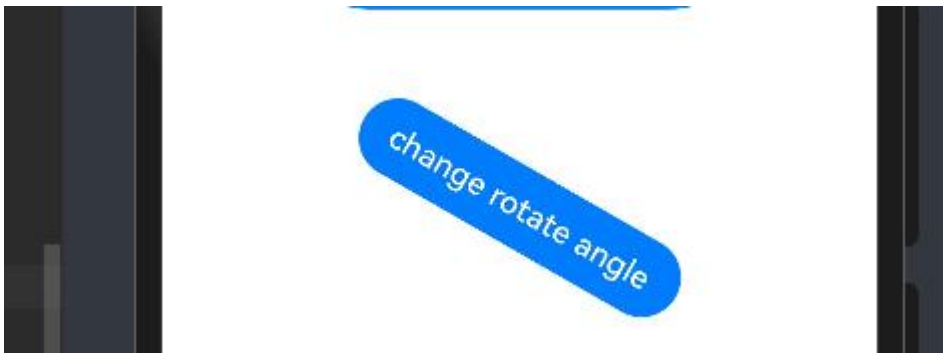
相信这个逻辑大家很容易理解。效果就是再次点击之后，按钮恢复到原来的大小。注意这里animateTo({}, ()...)中的{}是空的，也就是没有设置任何动画的属性参数，系统是使用默认的值。

再给第二个按钮添加点击事件：

```
        .onClick(() => {
          animateTo({
            duration: 1200,
            curve: Curve.Friction,
            delay: 500, // 动画延迟播放时间，单位为 ms(毫秒)，默认不延时播放
            iterations: -1, // 设置-1 表示动画无限循环
            playMode: PlayMode.Alternate,
            onFinish: () => {
              console.info('play end')
            },
            expectedFrameRateRange: { // 设置动画的期望帧率
              min: 10,
              max: 120,
              expected: 60,
            }
          }, () => {
            this.rotateAngle = 90
```

```
        })
    })
```

注意这里多了几个参数，效果：




最大角度到 90 度。

关于 Curve，参考这里：

https://developer.huawei.com/consumer/cn/doc/harmonyos-references-V5/js-apis-curve-V5#curve

**系统能力：** SystemCapability.ArkUI.ArkUI.Full

| 名称 | 说明 |
| --- | --- |
| Linear | 表示动画从头到尾的速度都是相同的。 |
| Ease | 表示动画以低速开始，然后加快，在结束前变慢，cubic-bezier(0.25, 0.1, 0.25, 1.0)。 |
| EaseIn | 表示动画以低速开始，cubic-bezier(0.42, 0.0, 1.0, 1.0)。 |
| EaseOut | 表示动画以低速结束，cubic-bezier(0.0, 0.0, 0.58, 1.0)。 |
| EaseInOut | 表示动画以低速开始和结束，cubic-bezier(0.42, 0.0, 0.58, 1.0)。 |
| FastOutSlowIn | 标准曲线，cubic-bezier(0.4, 0.0, 0.2, 1.0)。 |
| LinearOutSlowIn | 减速曲线，cubic-bezier(0.0, 0.0, 0.2, 1.0)。 |
| FastOutLinearIn | 加速曲线，cubic-bezier(0.4, 0.0, 1.0, 1.0)。 |
| ExtremeDeceleration | 急缓曲线，cubic-bezier(0.0, 0.0, 0.0, 1.0)。 |
| Sharp | 锐利曲线，cubic-bezier(0.33, 0.0, 0.67, 1.0)。 |
| Rhythm | 节奏曲线，cubic-bezier(0.7, 0.0, 0.2, 1.0)。 |
| Smooth | 平滑曲线，cubic-bezier(0.4, 0.0, 0.4, 1.0)。 |
| Friction | 阻尼曲线，cubic-bezier(0.2, 0.0, 0.2, 1.0)。 |

## 3. 属性动画-animation 属性方法

理论讲解部分参考：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-attribute-animation-overview-V5

注意这段话：

相比于 animateTo 接口需要把要执行动画的属性的修改放在闭包中，animation 接口无需使用闭包，把 animation 接口加在要做属性动画的可动画属性后即可。animation 只要检测到其绑定的可动画属性发生变化，就会自动添加属性动画，animateTo 则必须在动画闭包内改变可动画属性的值从而生成动画。

我们用相同的实例来演示。创建一个新的文件：PropertyAnimate2.ets

添加代码：

```
@Entry
@Component
struct PropertyAnimate2 {
  @State widthSize: number = 250
  @State heightSize: number = 100
  @State rotateAngle: number = 0
  @State flag: boolean = true

  build() {
    Column() {
      Button('change size')
        .onClick(() => {
```

```
            if (this.flag) {
                this.widthSize = 150
                this.heightSize = 60
            } else {
                this.widthSize = 250
                this.heightSize = 100
            }
            this.flag = !this.flag
        })
        .margin(30)
        .width(this.widthSize)
        .height(this.heightSize)
        .animation({
            duration: 2000,
            curve: Curve.EaseOut,
            iterations: 3,
            playMode: PlayMode.Normal
        })
    Button('change rotate angle')
        .onClick(() => {
            this.rotateAngle = 90
        })
```

```
            .margin(50)
            .rotate({ angle: this.rotateAngle })
            .animation({
              duration: 1200,
              curve: Curve.Friction,
              delay: 500,
              iterations: -1, // 设置-1 表示动画无限循环
              playMode: PlayMode.Alternate,
              expectedFrameRateRange: {
                min: 20,
                max: 120,
                expected: 90,
              }
            })
        }.width('100%').margin({ top: 20 })
      }
    }
```

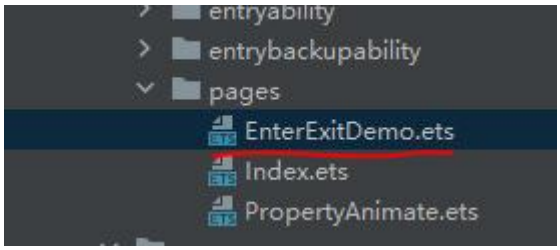
效果和 animateTo 是一样的。请注意这里的 onClick 事件中的代码，以及通过调用属性函数.animation 的方法的代码。比较和 animateTo 的使用方法的区别。

### 4. 出现/消失 转场动画

理论讲解：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-enter-exit-transition-V5

创建一个新的文件：EnterExitDemo.ets



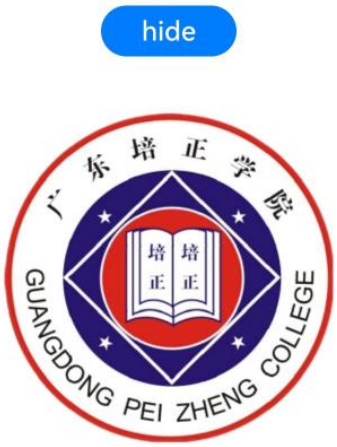

添加基础代码：

```
@Entry
@Component
struct EnterExitDemo {
  @State flag: boolean = true;
  @State control: string = 'hide';

  build() {
    Column() {
      Button(this.control).width(80).height(30).margin(30)
        .onClick(() => {
          // 点击 Button 控制 Image 的显示和消失
          if (this.flag) {
            this.control = 'show';
          } else {
            this.control = 'hide';
          }
          this.flag = !this.flag;
```

```
        })
        if (this.flag) {
            Image($r('app.media.laurence')).width(200).height(200)
        }
    }.width('100%')
  }
}
```

Previewer 效果：




点击“hide”按钮，图片消失。

我们给图片加上 transition 属性函数（红色字体部分）：

```
        if (this.flag) {
            // Image 的显示和消失配置为相同的过渡效果（出现和消失互为逆过程）
            Image($r('app.media.laurence')).width(200).height(200)
                .transition(TransitionEffect.OPACITY
                    .animation({ duration: 2000, curve: Curve.Ease })
```

```
            .combine(TransitionEffect.rotate({ z: 1, angle: 180 }))
        )
    }
```

这个 transition 里面的含义为：

- 出现时从指定的透明度为 0、绕 z 轴旋转 180°的状态，变为默认的透明度为 1、旋转角为 0 的状态，透明度与旋转动画时长都为 2000ms
- 消失时从默认的透明度为 1、旋转角为 0 的状态，变为指定的透明度为 0、绕 z 轴旋转 180°的状态，透明度与旋转动画时长都为 2000ms

效果：（这里只是中间状态的一个截图）


请自行尝试，查看效果。

再看另外一个实例，创建文件 EnterExitDemo2.ets，加入代码：

```
import { curves } from '@kit.ArkUI';


@Entry
@Component
struct TransitionEffectDemo {
```

```
  @State isPresent: boolean = false;


  // 第一步，创建 TransitionEffect
  private effect: TransitionEffect =
    // 创建默认透明度转场效果,并指定了 springMotion(0.6, 0.8)曲线
    TransitionEffect.OPACITY.animation({ curve: curves.springMotion(0.6, 0.8) })
      // 通过 combine 方法,这里的动画参数会跟随上面的 TransitionEffect，也就是springMotion(0.6, 0.8)
      .combine(TransitionEffect.scale({ x: 0, y: 0 }))
        // 添加旋转转场效果,这里的动画参数会跟随上面带 animation 的 TransitionEffect,也就是 springMotion(0.6, 0.8)
      .combine(TransitionEffect.rotate({ angle: 90 }))
        // 添加平移转场效果，这里的动画参数使用指定的 springMotion()
      .combine(TransitionEffect.translate({ y: 200 }).animation({ curve: curves.springMotion() }))
        // 添加 move 转场效果，这里的动画参数会跟随上面的 TransitionEffect，也就是springMotion()
      .combine(TransitionEffect.move(TransitionEdge.END))


  build() {
    Stack() {
      if (this.isPresent) {
```

```
            Column() {
                Text('ArkUI')
                    .fontWeight(FontWeight.Bold)
                    .fontSize(20)
                    .fontColor(Color.White)
            }
            .justifyContent(FlexAlign.Center)
            .width(150)
            .height(150)
            .borderRadius(10)
            .backgroundColor(0xf56c6c)
            // 第二步：将转场效果通过 transition 接口设置到组件
            .transition(this.effect)
        }

        // 边框
        Column()
            .width(155)
            .height(155)
            .border({
                width: 5,
                radius: 10,
```

```
            color: Color.Black
        })

        // 第三步：新增或者删除组件触发转场，控制新增或者删除组件
        Button('Click')
          .margin({ top: 320 })
          .onClick(() => {
            this.isPresent = !this.isPresent;
          })
      }
      .width('100%')
      .height('60%')
    }
  }
```

代码的核心是红色字体部分，通过用 TransitionEffect 对象进行动画的配置。具体的讲解参考：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-enter-exit-transition-V5

### 5. 动画曲线

理论讲解：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-curve-overview-V5

根据动画曲线是否符合物理世界客观规律，可将其分为物理曲线（ArkUI 当前提供了多种物理弹簧曲线）和传统曲线两种类型。相比于传统曲线，物理曲线产生的运动轨迹更加符合用户认知，有助于创造自然生动的动画效果，建议开发者优先使用物理曲线。

**先看一下传统曲线**。创建一个新的文件：CurvesDemo1.ets

添加基础代码：

```
@Entry
@Component
export struct CurvesDemo1 {

  build() {
    Column() {
      Stack() {
        // 摆动管道
        Row()
          .width(290)
          .height(290)
          .border({
            width: 15,
            color: 0xE6E8EB,
            radius: 145
          })

      }
      .width('100%')
      .height(200)
      .margin(100)
```

```
    }
    .width('100%')
  }
}
```

效果：

我们通过让小球在这个圆形轨道上转动来查看动画曲线的效果。

定义一个 MyCurve 类：

```
class MyCurve {
  public title: string;
  public curve: Curve;
  public color: Color | string;
  public text: string;

  constructor(title: string, curve: Curve, color: Color | string = '', text: string) {
    this.title = title;
```

```
    this.curve = curve;
    this.color = color;
    this.text = text;
  }
}
```

以及一个数组：

```
const myCurves: MyCurve[] = [
  new MyCurve(' Linear', Curve.Linear, '#317AF7', '1'),
]
```

这里代码的核心要点是用到了 Curve，首先在数组中放一个数据，曲线设置为 Linear，线性，也就是说动画的效果是线性变化的。

在 Stack()上面添加一个 Grid()容器，用来显示圆点：

```
      // 曲线图例
      Grid() {
        ForEach(myCurves, (item: MyCurve) => {
          GridItem() {
            Column() {
              Text(item.text)
                .width(30)
                .height(30)
                .borderRadius(15)
                .backgroundColor(item.color)
```

```
                .fontColor(Color.White)
                .textAlign(TextAlign.Center)
            Text(item.title)
                .fontSize(15)
                .fontColor(0x909399)
        }
        .width('100%')
    }
  })
}
.columnsTemplate('1fr 1fr 1fr')
.rowsTemplate('1fr 1fr 1fr 1fr 1fr')
.padding(10)
.width('100%')
.height(300).margin({top:50})
```

效果：


在 build()前面加上一个旋转角度的状态变量：

```
@State dRotate: number = 0; // 旋转角度
```

在 Stack()容器里面，摆动管道代码的下面加上：

```
ForEach(myCurves, (item: MyCurve) => {
// 小球
Column() {
Text(item.text)
.width(30)
.height(30)
.borderRadius(15)
.backgroundColor(item.color)
.textAlign(TextAlign.Center)
```

```
                .fontColor(Color.White)
            }
            .width(20)
            .height(300)
            .rotate({ angle: this.dRotate })
            .animation({ duration: 2000, iterations: -1, curve: item.curve, delay: 100 })
        })
```

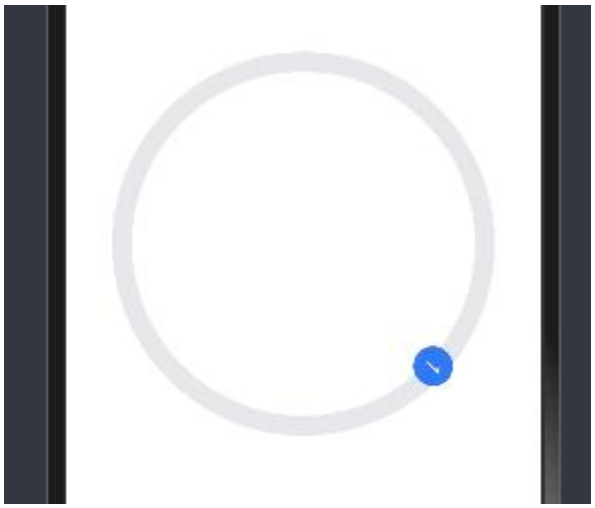
核心是，对于每一个圆点，旋转一个角度（dRotate 定义），无限循环（-1）。

对于整个 Stack 容器，加上点击事件，去改变它的值：

```
    .onClick(() => {
        this.dRotate ? null : this.dRotate = 360;
    })
```

这里是让其旋转 360 度。

刷新 Previewer，效果：

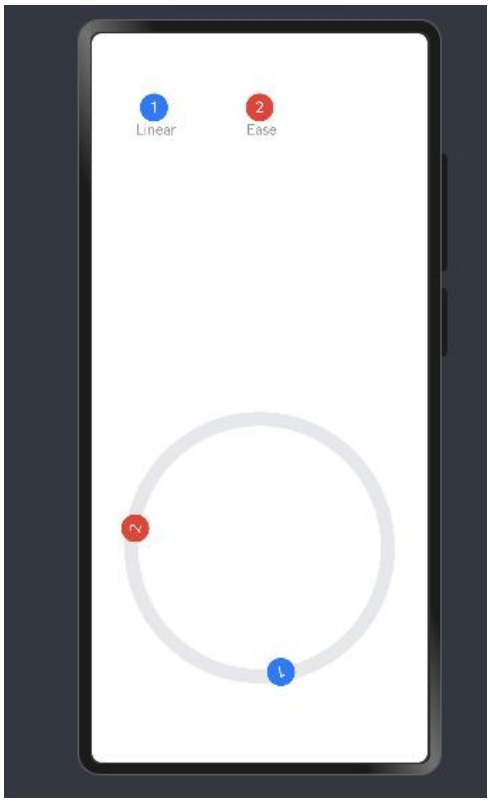


可以看到，圆点不断在旋转，速度是匀速的，线性的。

在数组里面再加一个数据：

```
new MyCurve(' Ease', Curve.Ease, '#D94838', '2'),
```

刷新后效果：




可以比较线性 Linear 和 Ease 的区别。

逐步增加数据：

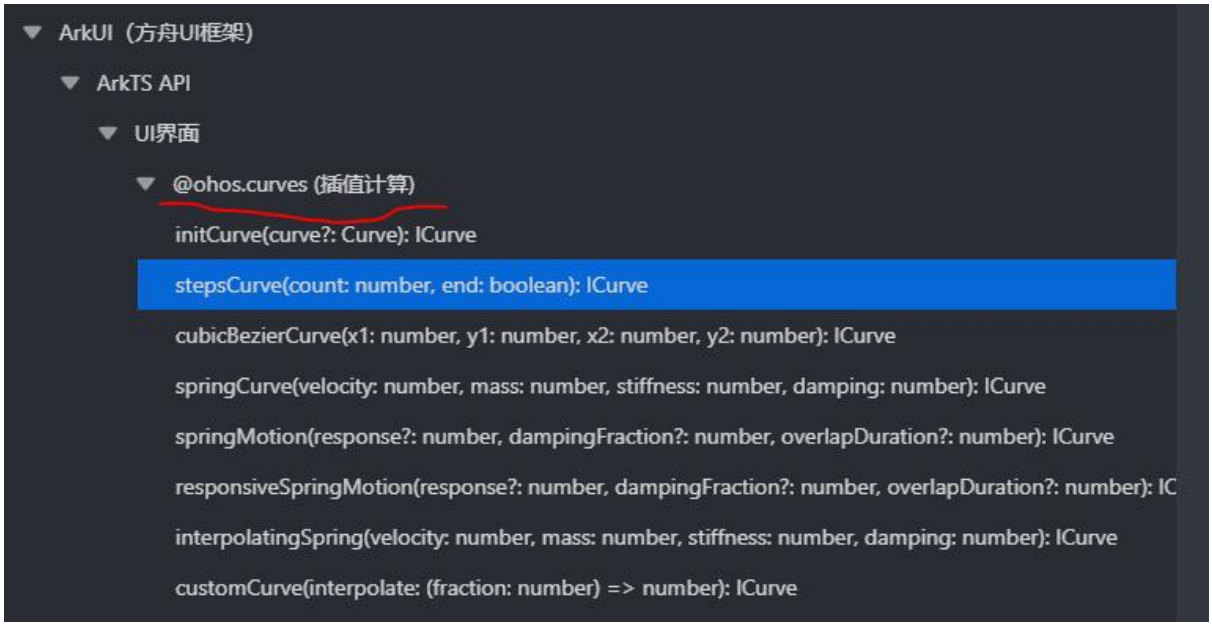
```
new MyCurve(' EaseIn', Curve.EaseIn, '#DB6B42', '3'),
new MyCurve(' EaseOut', Curve.EaseOut, '#5BA854', '4'),
new MyCurve(' EaseInOut', Curve.EaseInOut, '#317AF7', '5'),
new MyCurve(' FastOutSlowIn', Curve.FastOutSlowIn, '#D94838', '6')
```

最后的效果：


**再看一下弹簧曲线**，理论讲解：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-spring-curve-V5

在 API Reference 中：



创建一个新的文件 CurvesDemo2.ets，加入代码：

```
import { curves } from '@kit.ArkUI';
class Spring {
  public title: string;
  public subTitle: string;
  public iCurve: ICurve;

  constructor(title: string, subTitle: string, iCurve: ICurve) {
    this.title = title;
    this.iCurve = iCurve;
    this.subTitle = subTitle;
  }
}
@Entry
@Component
export struct SpringCurve {
  build() {
  }
}
```

定义了一个 Spring 类，核心是用到了 ICurve，曲线的插值对象:

| 类型 | 说明 |
| --- | --- |
| ICurve | 曲线对象。<br><br>**说明:**<br><br>弹性动画曲线为物理曲线，animation、animateTo、pageTransition中的duration参数不生效，动画持续时间取决于springMotion动画曲线参数和之前的速度。时间不能归一，故不能通过该曲线的interpolate函数获得插值。 |

再定义一个弹簧组件：

```
// 弹簧组件
@Component
struct Motion {
  @Prop dMove: number = 0
  private title: string = ""
  private subTitle: string = ""
  private iCurve: ICurve | undefined = undefined

  build() {
    Column() {
      Circle()
        .translate({ y: this.dMove })
        .animation({ curve: this.iCurve, iterations: -1 })
        .foregroundColor('#317AF7')
        .width(30)
        .height(30)
```

```
        Column() {
          Text(this.title)
            .fontColor(Color.Black)
            .fontSize(10).height(30)
          Text(this.subTitle)
            .fontColor(0xcccccc)
            .fontSize(10).width(50)
        }
        .borderWidth({ top: 1 })
        .borderColor(0xf5f5f5)
        .width(80)
        .alignItems(HorizontalAlign.Center)
        .height(100)

      }
      .height(110)
      .margin({ bottom: 5 })
      .alignItems(HorizontalAlign.Center)
    }
  }
```

核心代码为红色字体，我们画出一个圆点，在垂直方向 y 方向移动一定的距离，然后动画的效果是遵循 iCurve 定义的曲线模式。

@Entry 部分的代码改为：

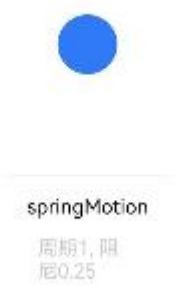
```
@Entry
@Component
export struct SpringCurve {
  @State dMove: number = 0;
  private springs: Spring[] = [
    new Spring('springMotion', '周期 1，阻尼 0.25', curves.springMotion(1, 0.25)),
  ];

  build() {
    Row() {
      ForEach(this.springs, (item: Spring) => {
        Motion({ title: item.title, subTitle: item.subTitle, iCurve: item.iCurve, dMove: this.dMove })
      })
    }
    .justifyContent(FlexAlign.Center).alignItems(VerticalAlign.Bottom)
    .width('100%')
    .height(437)
    .margin({ top: 20 })
    .onClick(() => {
      this.dMove = -50;
```

```
    })
  }
}
```

就是说，我们先尝试 SpringMotion 的效果，API Reference:


效果：

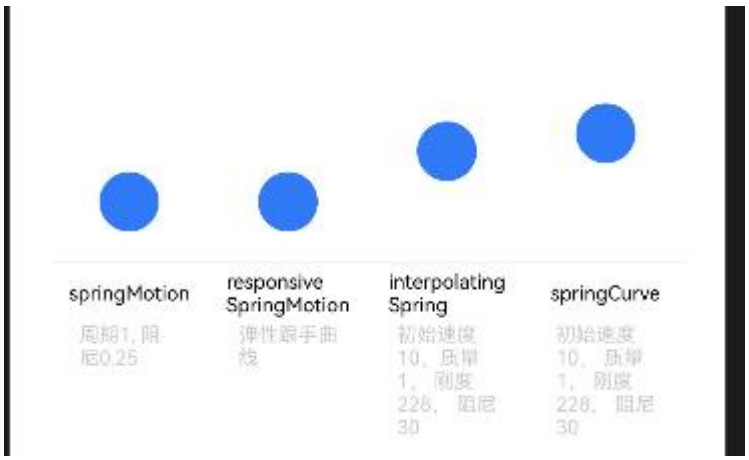



圆球弹起之后，像弹簧的效果，逐步上下移动最后停留在-50（dMove）的位置。

逐步添加另外几个到数组中：

```
    new Spring('responsive' + '\n' + 'SpringMotion', '弹性跟手曲线', curves.responsiveSpringMotion(1, 0.25)),
    new Spring('interpolating' + '\n' + 'Spring', '初始速度 10，质量 1， 刚度 228， 阻尼 30', curves.interpolatingSpring(10, 1, 228, 30)),
    new Spring('springCurve', '初始速度 10， 质量 1， 刚度 228， 阻尼 30', curves.springCurve(10, 1, 228, 30))
```

效果：




截图其实看不出效果，请自己尝试查看效果。

### 6. 导航转场动画

理论讲解：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-navigation-transition-V5

导航转场是页面的路由转场方式，也就是一个界面消失，另外一个界面出现的动画效果。导航转场推荐使用 Navigation 组件实现，可搭配 NavDestination 组件实现导航功能。

由于这个是系统自己实现的动画，请直接尝试上面链接中的实例。注意，需要在模拟器上运行才可以。

## 五、实验注意事项

1. 注意教师的操作演示。
2. 学生机与教师机内网连通，能接收和提交实验结果。
3. 按实验要求输入测试数据，并查看输出结果是否符合实验结果。

## 六、思考题

1. 通过这个实验，你学到了什么？