# 2.1 动画

流畅、有意思的动画是 VR Web 用户体验的基础。

## Animated

Animated 库提供了多样性的动画，使得互动模式更高效。

Animated 专注于基于时间动画的开始和结束方法之间变换的输入输出关系。

举个例子：下面是一个简单的弹跳动画组件

```
class Playground extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      bounceValue: new  Animated.Value(0),
    };
  }
  render() {
    return (
      <Animated.Image                    // Base: Image, Text,
View
        <source={{uri: 'http://i.imgur.com/XMKOH81.jpg'}}
        <style={{
          flex: 1,
          width: 1,
          height: 1,
          transform: [                    // `transform` is an
ordered array
```

```
            {scale: this.state.bounceValue},         // Map
`bounceValue` to `scale`
          ]
        }}
      />
    );
  }
  componentDidMount() {
    this.state.bounceValue.setValue(1.5);                    //
Start large
    Animated.spring(                        // Base: spring, decay,
timing
      this.state.bounceValue,               // Animate `bounceValue`
      {
        toValue: 0.8,                        // Animate to smaller
size
        friction: 1,                        // Bouncier spring
      }
    ).start();                              // Start the animation
  }
}
```

上面的例子中，`bounceValue` 在构造函数的 `state` 就初始化了，图片的 `scale` 变换取得就是这个

值，在组件加载完毕后，这个值变成了 1.5，开始一个弹跳的动画。

## Core API

大多数情况你都可以直接用 `Animated` ：

- 两个值类型： 单个值用`Value`，两个值得用`ValueXY`

- 三个动画类型：`spring`、`decay`、`timing`
- 三个组件类型：`View`、`Text`、`Image`

其他组件用 `Animated.createAnimatedComponent`

三个动画类型可以创建出几乎所有的动画曲线：

- **`spring`**：符合 Origami 的简单的单次弹跳物理模型
  - **`friction`**：简单理解就是弹跳的高度，默认是 7
  - **`tension`**：这是控制速度的，默认值是 40
- **`decay`**：开始有个初速度，然后逐渐减慢直到停止
  - **`velocity`**：初始速度：这是必有的值
  - **`deceleration`**：减速度，默认值是 0.997
- **`timing`**：从时间范围映射到渐变的值
  - **`duration`**：动画的长度（ms）默认值 500
  - **`easing`**：定义曲线的缓动函数，查看 **`Easing`** 模块可以看到许多预定义的函数，IOS 默认的是

```
Easing.inOut(Easing.ease).
```

  - **`delay`**：延时多长时间再启动动画（ms）默认值 0

调用 **`start()`** 开启动画,动画结束回调 **`start()`**，如果动画正常执行完毕，回调的结果是

**`{finished: true}`**,但是如果是调用了 **`stop()`** 结束的动画（比如：动画被手势或者其他的动画打断

了），回调的结果就是 **{finished: false}** 。

## Composing Animations(组合动画)

动画可以把 `parallel sequence stagger delay` 组合起来，每一个组合动画都接受一个数组，并在适

当的时候调用 **start、stop** ，例如：

```
Animated.sequence([           // spring to start and twirl after
decay
finishes
  Animated.decay(position, {           // coast to a stopf
    velocity: {x: gestureState.vx, y: gestureState.vy},   //
velocity from gesture
release
    deceleration: 0.997,
  }),
  Animated.parallel([           // after decay, in parallel:
    Animated.spring(position, {
      toValue: {x: 0, y: 0.}      // return to start
    }),
    Animated.timing(twirl, {           // and twirl
      toValue: 360,
    }),
  ]),
]).start();           // start the sequence group
```

默认情况下，如果任何一个动画被停止或者中断了，组内的所有其他的动画也会被停止，Parallel 有

一个 **stopTogether** 属性，如果设置为 **false:** ，可以禁用自动停止。

## Interpolation (插值)

`Animates` API 还有一个很强大的部分就是 `interpolation` 插值函数，他可以接受一个输入区间，然

后映射到另一个的输出区间，下面是一个简单 0-1 区间到 0-100 区间的映射示例：

```
value.interpolate({
  inputRange: [0, 1],
  outputRange: [0, 100],
}),
```

`interpolation` 还支持多个区间段落，长用来定义精致区间等。举个例子，要让输入在进入-300 时

取相反值，然后在输入接近-100 时到达 0，然后在输入接近 0 时又回到 1，接着一直到输入 100 的过程

中逐步回到 0，最后形成一个保持在 0 上：

```
value.interpolate({
  inputRange: [-300, -100], 0], 100], 101]
  inputRange: [300, 0], 1], 0], 0]
}),
```

最终映射的结果如下：

输入 输出

-400 450

-300 300

-200 150

-100 0

-50 0.5

0 1

50 0

100 0

101 0

200 0

**interpolation** 还支持映射到字符串，你可以选择颜色的映射，也可以映射成旋转的角度等等：

```
value.interpolate({
  inputRange: [0, 360],
  outputRange: ['0deg', '360deg'],
}),
```

**interpolation** 还支持任意的渐变函数，其中有很多已经在 **Easing** 中定义了，包括二次、指数、贝

塞尔曲线以及 **step** 、 **bounce** 等方法， **interpolation** 还支持限制输出区间 **outputRange** 。我们可

以通过设置 **extrapolate** 、 **extrapolateLeft** 、 **extrapolateRight** 属性来限制输出区间，默认值是

**extend** （允许超出），但是我们可以使用 **clamp** 选项来组织输出值超过 **outputRange** 。

## Tracking Dynamic Values（跟踪动态值）

动画中所设的值还尅通过跟踪别的值得到，我们只要把 **toValue** 设置成另一个动态值，而不是一个

普通的数字。比如我们可以用弹跳动画来实现聊天头像的闪动，还可通过 **timing** 设置 **duration:0**

来实现快速跟踪，还可以用插值来进行组合：

```
Animated.spring(follower, {toValue: leader}).start();
Animated.timing(opacity, {
  toValue: pan.x.interpolate({
    inputRange: [0, 300],
    outputRange: ['1', '0'],
  }),
}).start();
```

**ValueXY** 是一个处理 2D 交互的方法，如旋转拖拽等，它是一个简单的包含了 **Animated.Value** 实例的

包装，然后提供了一系列辅助函数，是的 **ValueXY** 在许多时候可以替代 **Value** 来使用，比如上面的

代码片段中， **leader** 、 **follower** 可以同时为 **ValueXY** 类型，这样 x、y 的值都会被跟踪的。

## Responding to the Current Animation Value(响应当前的动画值)

你可能发现在动画执行的过程中，没有明显的方法可以拿到动画的值，出于优化考虑的，只有原生

代码运行阶段才知道，如果你想在 js 中也得到当前的动画值，下面有两个方法：

- `spring.stopAnimation(callback)` 停止动画，回调一个值回来，这在处理手势动画的时候有用
- `spring.addListener(callback)` 在动画执行的过程中持续异步调用回调函数，提供一个最接近

的值作为参数，这在用于触发状态切换的时候有用，如当用户拖拽一个东西靠近的时候弹出一个

新的气泡选项，不过这个状态切换可能并不会十分灵敏，因为它不像许多连续手势操作（旋转）

那样在 60fps 下运行

## requestAnimationFrame(请求帧动画)

`requestAnimationFrame` 在浏览器上用的比较多，接受一个参数，在下次绘制之前调用这个函数，一

些基于 JS 的动画库高度依赖这个 API，一般来说我们不用自己调用它，动画库会帮你管理好的

### setNativeProps 备注

**setNativeProps** 允许我们修改原生组件的属性，原生组件不像自己合成的组件，不用 **setState** 和重新渲染组件。我们可以用这来更新 scale，这对高度嵌套和没有使用 **shouldComponentUpdate** 优化过的组件是很有用的

```
// Back inside of the App component, replace the scrollSpring
listener

// in componentWillMount with this:

this._scrollSpring.addListener({

  onSpringUpdate: () => {

    if (!this._photo) {return }

    var v = this._scrollSpring.getCurrentValue();

    var newProps = {style: {transform: [{scaleX: v}, {scaleY:
v}]}};

    this._photo.setNativeProps(newProps);

  },

}),



// Lastly, we update the render function to no longer pass in the

// transform via style (avoid clashes when re-rendering) and to
set the

// photo ref

render() {

  return (

    <View style={styles.container}>

      <TouchableWithoutFeedback onPressIn={this._onPressIn}
onPressOut={this._onPressOut}>

        <Image ref={component => this._photo = component}

            source={{uri: "img/ReboundExample.png"}}

            style={{width: 250, height: 200}}
```

```
      </TouchableWithoutFeedback>
    </View>
  );
}
```

react-tween-state 是一个极小的库，在 react-tween-state 库里面用 **setNativeProps** 没有意义

因为这个库已经处理好了.

Rebound，另一种动画库，它在概念上类似 react-tween-state：你有一个起始值，然后定义一个

结束值，然后 Rebound 会生成所有中间的值并用于你的动画，用 **onSpringUpdate** 函数可得到每一帧

的值.更过的介绍在 React Native 上有详细的介绍：Animations

如果发生丢帧的情况，可以用 **setNativeProps** 、 **shouldComponentUpdate** 优化.

## LayoutAnimation(布局动画)

**LayoutAnimation** 允许你在全局范围内创建和更新动画，这些动画会在下一次渲染或布局周期运

行。它常用来更新 flexbox 布局，因为它可以无需测量或者计算特定属性就能直接产生动画。尤其是

当布局变化可能影响到父节点

譬如“查看更多”展开动画既增加父节点的尺寸又会将位于本行之下的所有行向下推动时，如果不使

用 LayoutAnimation，可能就需要显式声明组件的坐标，才能使得所有受影响的组件能够同步运行

动画。

注意尽管 LayoutAnimation 非常强大且有用，但它对动画本身的控制没有 Animated 或者其它动画库

那样方便，所以如果你使用 LayoutAnimation 无法实现一个效果，那可能还是要考虑其他的方案。

更多信息查看 LayoutAnimation.js