Version: 3.x

Glossary of terms

Animated component

Animated components are components which Reanimated can animate. Reanimated comes with just a handful of built-in components like `Animated.View`, `Animated.ScrollView`, or `Animated.ScrollView`.

```
import Animated from 'react-native-reanimated';

function App() {
  return (
    <Animated.View
      style={{
        width: 100,
        height: 100,
        backgroundColor: 'violet',
      }}
    />
  );
}
```

For components which aren't a part of Reanimated, to make their props and styles animatable, you have to wrap them with `createAnimatedComponent`:

```
import Animated from 'react-native-reanimated';
import { Circle } from 'react-native-svg';

const AnimatedCircle = Animated.createAnimatedComponent(Circle);
```

Shared value

Shared values are a driving factor of all your animations in Reanimated.

They are defined using `useSharedValue` hook:

```
import { useSharedValue } from 'react-native-reanimated';
```

and accessed and modified by their `.value` property:

```
function App() {
  const sv = useSharedValue(0);

  const handlePress = () => {
    sv.value += 10;
  };

  // rest of your glamorous code ✨
}
```

It can be a value of any type. When used, the data stored in a shared value is automatically synchronized between the **JavaScript thread** and the **UI thread**.

Animatable value

An animatable value refers to a type of value that can be used for animations. These values include numbers, strings, and arrays of numbers.

String values can be animated if they are in a specific format, such as `"10deg"`, `"21%"`, or even colors like `"#ffaabb"` or `"rgba(100, 200, 100, 0.7)"`.

Animation function

Animation functions are functions which let you create animations. They are building blocks that describe how your animations should behave.

Reanimated comes with 3 built-in animation functions:

- `withSpring` lets you create spring-based animation
- `withTiming` lets you create an animation based on duration and **easing**
- `withDecay` lets you create animations that mimic objects in motion that move with a given deceleration rate

These functions can be combined with **animation modifiers** to create rich and complex animations.

Animation modifier

Animation modifiers are functions used to customize animations.

Reanimated comes with 3 built-in modifiers:

- `withDelay` lets you add a delay before the animation starts
- `withRepeat` lets you repeat an animation certain number of times
- `withSequence` lets you chain animation one after the other

Animation object

An animation object is a value returned from **animation functions** and **modifiers** which holds the current state of the animation including its start and end conditions, as well as a `onFrame` function. These values allow to calculate the animation state for each frame.

When you pass an animation object to a **shared value** it is automatically treated as an **animatable value**.

For example, consider this code:

```
sv.value = withSpring(100);
```

Even though, the `withSpring` returns an animation object the final result that is stored in a shared value is just a number.

Animations in inline styling

Passing shared values directly to `style` property without the use of `useAnimatedStyle`.

For example:

```
function App() {
  const width = useSharedValue(100);

  return <Animated.View style={{ width }} />;
}
```

Worklet

Worklets are short-running JavaScript functions that can be run on the **UI thread**. They can also be run on a JavaScript thread just as you would run a function in your code.

Most of the time when working with Reanimated the code is automatically **workletized** and run on the UI thread by default.

```
const style = useAnimatedStyle(() => {
  console.log('Running on the UI thread');
  return { opacity: 0.5 };
});
```

You can create your own worklets using the `"worklet";` directive at the top of a function.

```
function myWorklet() {
  'worklet';
  console.log('Running on the UI thread');
}
```

`runOnUI` lets you manually run worklets on the UI thread:

```
function myWorklet(greeting) {
  'worklet';
  console.log(`${greeting} from the UI thread`);
}

function onPress() {
  runOnUI(myWorklet)('Howdy');
}
```

to workletize

To convert a JavaScript function into a serializable object which can be copied and ran over on UI thread.

Functions marked with `"worklet";` directive are automatically picked up and workletized by the Reanimated Babel plugin.

JavaScript thread

JavaScript thread (or JS thread for short) is responsible for handling JavaScript code execution in the app.

This is the primary place where the React Native app code is executed.

UI thread

UI thread is responsible for handling user interface updates. Also known as Main thread.

You can learn more about it by reading the **Threading model** article in the official React Native docs.

Reanimated Babel plugin

The plugin performs automatic **workletization** of certain functions used with Reanimated to reduce the amount of boilerplate code.

You can learn the details by reading the **Reanimated Babel plugin README**.

Edit this page