

Timers

Timers are an important part of an application and React Native implements the browser timers.

Timers

- `setTimeout`, `clearTimeout`
- `setInterval`, `clearInterval`
- `setImmediate`, `clearImmediate`
- `requestAnimationFrame`, `cancelAnimationFrame`

`requestAnimationFrame(fn)` is not the same as `setTimeout(fn, 0)` - the former will fire after all the frames have flushed, whereas the latter will fire as quickly as possible (over 1000x per second on a iPhone 5S).

`setImmediate` is executed at the end of the current JavaScript execution block, right before sending the batched response back to native. Note that if you call `setImmediate` within a `setImmediate` callback, it will be executed right away, it won't yield back to native in between.

The `Promise` implementation uses `setImmediate` as its asynchronicity implementation.

NOTE

When debugging on Android, if the times between the debugger and device have drifted; things such as animation, event behavior, etc., might not work properly or the results may not be accurate. Please correct this by running `adb shell "date `date +%m%d%H%M%Y.%S%3N`"` on your debugger machine. Root access is required for the use in real device.

InteractionManager

One reason why well-built native apps feel so smooth is by avoiding expensive operations during interactions and animations. In React Native, we currently have a limitation that there is only a single JS execution thread, but you can use `InteractionManager` to make sure long-running work is scheduled to start after any interactions/animations have completed.

Applications can schedule tasks to run after interactions with the following:

```
InteractionManager.runAfterInteractions(() => {  
  // ...long-running synchronous task...  
});
```

Compare this to other scheduling alternatives:

- `requestAnimationFrame()`: for code that animates a view over time.
- `setImmediate/setTimeout/setInterval()`: run code later, note this may delay animations.
- `runAfterInteractions()`: run code later, without delaying active animations.

The touch handling system considers one or more active touches to be an 'interaction' and will delay `runAfterInteractions()` callbacks until all touches have ended or been cancelled.

`InteractionManager` also allows applications to register animations by creating an interaction 'handle' on animation start, and clearing it upon completion:

```
const handle = InteractionManager.createInteractionHandle();  
// run animation... (`runAfterInteractions` tasks are queued)  
// later, on animation completion:  
InteractionManager.clearInteractionHandle(handle);  
// queued tasks run if all handles were cleared
```

Is this page useful?  



Edit this page

*Last updated on **Jun 21, 2023***