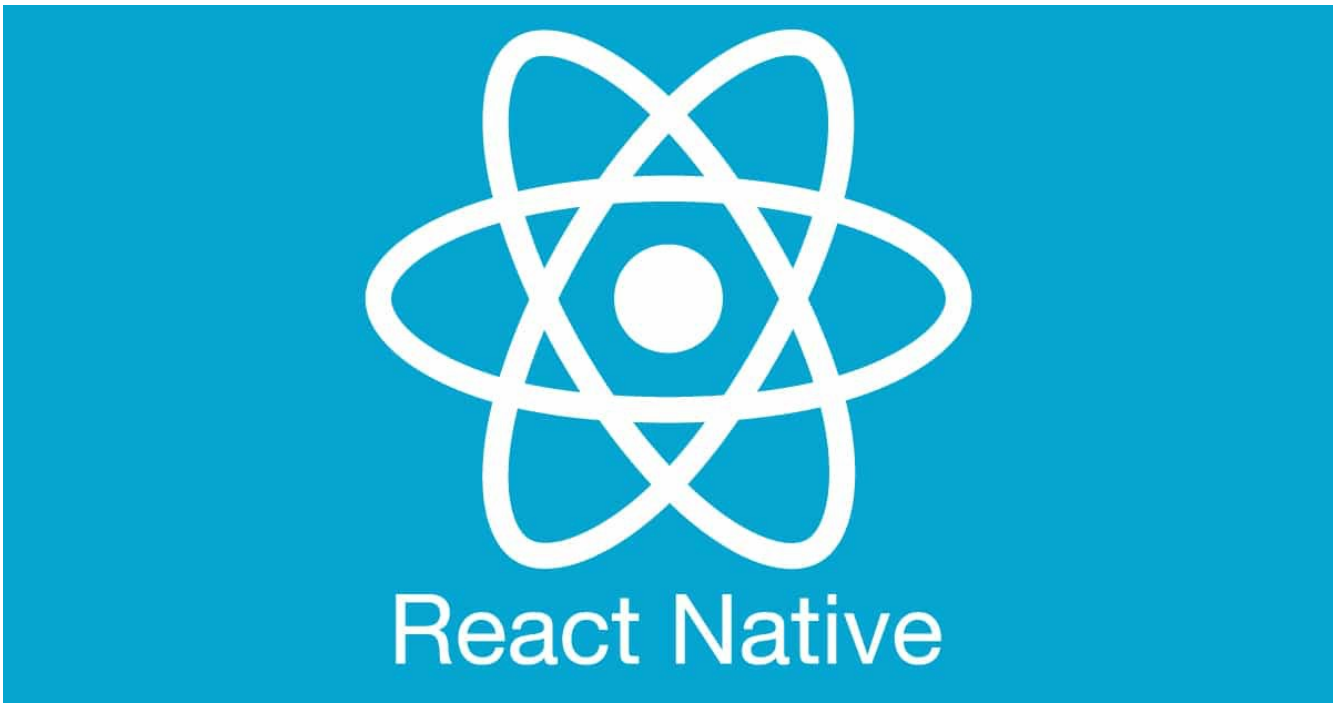


Documentación oficial React Native

1. Los básicos
2. Guías

Índice

Los básicos

Para comenzar

Esta página te ayudará a instalar y construir tu primera aplicación React Native. Si ya tiene React Native instalado, puede pasar directamente al Tutorial .

- Inicio rápido

La aplicación Create React Native es la forma más fácil de comenzar a construir una nueva aplicación React Native. Le permite iniciar un proyecto sin instalar ni configurar ninguna herramienta para compilar código nativo; no se requiere instalación de Xcode o Android Studio (consulte Advertencias).
Suponiendo que tiene Node instalado, puede usar npm para instalar la `create-react-native-app` utilidad de línea de comandos:

```
npm install -g create-react-native-app
```

Luego ejecute los siguientes comandos para crear un nuevo proyecto React Native llamado "AwesomeProject":

```
create-react-native-app AwesomeProject
cd AwesomeProject
npm start
```

Esto iniciará un servidor de desarrollo para usted e imprimirá un código QR en su terminal.

Ejecutando su aplicación React Native

Instale la aplicación del cliente Expo en su teléfono iOS o Android y conéctese a la misma red inalámbrica que su computadora. Con la aplicación Expo, escanee el código QR de su terminal para abrir su proyecto.

Modificando su aplicación

Ahora que ha ejecutado con éxito la aplicación, modifiquémosla. Abra `App.js` en su editor de texto de su elección y edite algunas líneas. La aplicación debe volver a cargarse automáticamente una vez que guarde los cambios.

¡Eso es!

¡Felicitaciones! Has ejecutado y modificado satisfactoriamente tu primera aplicación React Native.

¿Ahora que?

- La aplicación Create React Native también tiene una guía de usuario a la que puede hacer referencia si tiene preguntas específicas para la herramienta.
- Si no puede hacer que esto funcione, consulte la sección Solución de problemas en la aplicación README for Create React Native.

Si tiene curiosidad por saber más sobre React Native, continúe con el Tutorial .

Ejecutando su aplicación en un simulador o dispositivo virtual

La aplicación Create React Native hace que sea muy fácil ejecutar su aplicación React Native en un dispositivo físico sin configurar un entorno de desarrollo. Si desea ejecutar su aplicación en el simulador de iOS o en un dispositivo virtual de Android, consulte las instrucciones para crear proyectos con código nativo para aprender a instalar Xcode y configurar su entorno de desarrollo de Android.

Una vez que haya configurado esto, puede ejecutar su aplicación en un dispositivo virtual Android ejecutando `npm run android` o ejecutando el simulador iOS `npm run ios` (solo macOS).

Advertencias

Debido a que no crea ningún código nativo al usar la aplicación Create React Native para crear un proyecto, no es posible incluir módulos nativos personalizados más allá de las API y componentes de React Native que están disponibles en la aplicación del cliente de Expo.

Si sabe que eventualmente necesitará incluir su propio código nativo, la aplicación Create React Native todavía es una buena manera de comenzar. En ese caso, solo tendrá que " expulsar " eventualmente para crear sus propias compilaciones nativas. Si expulsa, se requerirán las instrucciones "Construir proyectos con código nativo" para continuar trabajando en su proyecto.

Create React Native App configura su proyecto para usar la versión más reciente de React Native que es compatible con la aplicación del cliente de Expo. La aplicación de cliente de Expo generalmente gana soporte para una versión de React Native dada aproximadamente una semana después de que la versión de React Native se publique como estable. Puede consultar este documento para averiguar qué versiones son compatibles.

Si está integrando React Native en un proyecto existente, querrá omitir la aplicación Create React Native y acceder directamente a la configuración del entorno de compilación nativo. Seleccione "Creación de proyectos con código nativo" más arriba para obtener instrucciones sobre la configuración de un entorno de compilación nativo para React Native.

Aprenda los conceptos básicos

React Native es como React, pero usa componentes nativos en lugar de componentes web como bloques de construcción. Entonces, para comprender la estructura básica de una aplicación React Native, necesita comprender algunos de los conceptos básicos de React, como JSX, componentes `state`, y `props`. Si ya conoces Reaccionar, aún necesitas aprender algunas cosas específicas de React-Native, como los componentes nativos. Este tutorial está dirigido a todos los públicos, ya sea que tenga o no experiencia en React.
Hagamos esto.

Hola mundo

De acuerdo con las antiguas tradiciones de nuestra gente, primero debemos construir una aplicación que no haga más que decir "Hola mundo". Aquí está:

```
import React, { Component } from 'react';
import { Text } from 'react-native';

export default class HelloWorldApp extends Component {
  render() {
    return (
      <Text>Hello world!</Text>
    );
  }
}
```

Si siente curiosidad, puede jugar con código de muestra directamente en los simuladores web. También puede pegarlo en su `App.js` archivo para crear una aplicación real en su máquina local.

¿Que está pasando aqui?

Algunas de las cosas aquí podrían no parecerse a JavaScript. No entres en pánico *Este es el futuro* .
En primer lugar, ES2015 (también conocido como ES6) es un conjunto de mejoras en JavaScript que ahora es parte del estándar oficial, pero que aún no es compatible con todos los navegadores, por lo que a menudo no se usa aún en el desarrollo web. Reacciona Navíos nativos con compatibilidad con ES2015, para que puedas usar esto sin preocuparte por la compatibilidad. `import`, `from`, `class`, `extends`, Y la `() =>`sintaxis en el ejemplo anterior son todas las características ES2015. Si no está familiarizado con ES2015, probablemente pueda obtenerlo simplemente leyendo el código de muestra que tiene este tutorial. Si lo desea, esta página tiene una buena descripción de las características de ES2015.
La otra cosa inusual en este ejemplo de código es `<Text>Hello world!</Text>`. Esta es JSX, una sintaxis para incrustar XML dentro de JavaScript. Muchos marcos utilizan un lenguaje de plantillas especial que le permite incrustar código dentro del lenguaje de marcado. En React, esto se invierte. JSX te permite escribir tu lenguaje de marcado dentro del código. Se ve como HTML en la web, excepto que en lugar de elementos web como , `<div>`o `<span>`, usa componentes Reaccionar. En este caso, `<Text>` es un componente incorporado que solo muestra algo de texto.

Componentes

Entonces este código está definiendo `HelloWorldApp`, es nuevo `Component`. Cuando construya una aplicación React Native, creará nuevos componentes mucho. Todo lo que ves en la pantalla es algún tipo de componente. Un componente puede ser bastante simple: lo único que se requiere es una `render` función que devuelva algo de JSX para representar.

Esta aplicación no hace mucho

Buen punto. Para hacer que los componentes hagan cosas más interesantes, debes aprender sobre Props .

Apoyos

La mayoría de los componentes se pueden personalizar cuando se crean, con diferentes parámetros. Estos parámetros de creación se llaman `props`.
Por ejemplo, un componente básico de React Native es el `Image`. Cuando crea una imagen, puede usar un accesorio designado `source` para controlar qué imagen muestra.

Observe que `{pic}` está rodeado de llaves para insertar la variable `pic` en JSX. Puedes poner cualquier expresión de JavaScript entre llaves en JSX.
Tus propios componentes también pueden usar `props`. Esto le permite crear un único componente que se usa en muchos lugares diferentes en su aplicación, con propiedades ligeramente diferentes en cada lugar. Solo refiérase a `this.props` en su `render` función. Aquí hay un ejemplo:

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Greeting extends Component {
  render() {
    return (
      <Text>Hello {this.props.name}!</Text>
    );
  }
}

export default class LotsOfGreetings extends Component {
  render() {
    return (
      <View style={{alignItems: 'center'}}>
        <Greeting name='Rexxar' />
        <Greeting name='Jaina' />
        <Greeting name='Valeera' />
      </View>
    );
  }
}

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => LotsOfGreetings);
```

El uso `name` como accesorio nos permite personalizar el `Greeting` componente, por lo que podemos reutilizar ese componente para cada uno de nuestros saludos. Este ejemplo también usa el `Greeting` componente en JSX,

al igual que los componentes integrados. El poder de hacer esto es lo que hace que React sea tan genial: si crees que deseas tener un conjunto diferente de primitivas UI con las que trabajar, simplemente inventas unas nuevas.
La otra cosa nueva que está sucediendo aquí es el `View` componente. A `View` es útil como contenedor para otros componentes, para ayudar a controlar el estilo y el diseño.
Con `props` y los componentes básicos `Text`, `Image` y `View`, puede construir una amplia variedad de pantallas estáticas. Para aprender cómo hacer que tu aplicación cambie con el tiempo, debes aprender sobre el estado .

Estado

Hay dos tipos de datos que controlan un componente: `props` y `state`. `Props` son establecidos por el padre y se arreglan a lo largo de la vida de un componente. Para los datos que van a cambiar, tenemos que usar `state`.
En general, debe inicializar `state` en el constructor y luego llamar `setState` cuando desee cambiarlo.
Por ejemplo, digamos que queremos hacer texto que parpadee todo el tiempo. El texto en sí se establece una vez cuando se crea el componente parpadeante, por lo que el texto en sí mismo es a `prop`. El "si el texto está actualmente activado o desactivado" cambia con el tiempo, por lo que debe mantenerse `state`.

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Blink extends Component {
  constructor(props) {
    super(props);
    this.state = {showText: true};

    // Toggle the state every second
    setInterval(() => {
      this.setState(previousState => {
        return { showText: !previousState.showText };
      });
    }, 1000);
  }

  render() {
    let display = this.state.showText ? this.props.text : ' ';
    return (
      <Text>{display}</Text>
    );
  }
}

export default class BlinkApp extends Component {
  render() {
    return (
      <View>
        <Blink text='I love to blink' />
        <Blink text='Yes blinking is so great' />
        <Blink text='Why did they ever take this out of HTML' />
        <Blink text='Look at me look at me look at me' />
      </View>
    );
  }
}
```

```
// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => BlinkApp);
```

En una aplicación real, probablemente no establecerá el estado con un temporizador. Puede establecer el estado cuando lleguen nuevos datos desde el servidor o desde la entrada del usuario. También puede usar un contenedor de estado como Redux para controlar su flujo de datos. En ese caso, usaría Redux para modificar su estado en lugar de llamar `setState` directamente.

Cuando se llama a setState, BlinkApp volverá a renderizar su Componente. Al llamar a setState dentro del temporizador, el componente volverá a reproducirse cada vez que el temporizador marque.

Estado funciona de la misma manera que en React, por lo que para obtener más detalles sobre el estado de manejo, puede mirar la API React.Component . En este punto, es posible que le moleste que la mayoría de nuestros ejemplos hasta el momento usen aburrido texto negro predeterminado. Para hacer las cosas más bellas, tendrás que aprender sobre Style .

Con React Native, no utiliza un lenguaje especial o sintaxis para definir estilos. Usted acaba de diseñar su aplicación usando JavaScript. Todos los componentes principales aceptan un apoyo nombrado `style`. Los nombres y valores de estilo generalmente coinciden con el funcionamiento de CSS en la web, excepto que los nombres se escriben usando mayúsculas y miniaturas, por ejemplo, en `backgroundColor` lugar de hacerlo `background-color`.

El `style`accesorio puede ser un simple objeto JavaScript antiguo. Eso es lo más simple y lo que generalmente usamos para el código de ejemplo. También puede pasar una serie de estilos: el último estilo de la matriz tiene prioridad, por lo que puede usar esto para heredar estilos.
A medida que un componente crece en complejidad, a menudo es más limpio usarlo `StyleSheet.create` para definir varios estilos en un solo lugar. Aquí hay un ejemplo:

```
import React, { Component } from 'react';
import { AppRegistry, StyleSheet, Text, View } from 'react-native';

export default class LotsOfStyles extends Component {
  render() {
    return (
      <View>
        <Text style={styles.red}>just red</Text>
        <Text style={styles.bigblue}>just bigblue</Text>
        <Text style={[styles.bigblue, styles.red]}>bigblue, then red</Text>
        <Text style={[styles.red, styles.bigblue]}>red, then bigblue</Text>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  bigblue: {
    color: 'blue',
    fontWeight: 'bold',
    fontSize: 30,
  },
  red: {
    color: 'red',
  },
});

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => LotsOfStyles);
```

Un patrón común es hacer que su componente acepte un `style` apoyo que a su vez se usa para diseñar subcomponentes. Puede usar esto para hacer que los estilos "en cascada" sean como lo hacen en CSS.
Hay muchas más formas de personalizar el estilo del texto. Consulte la referencia del componente de texto para obtener una lista completa.

Ahora puedes hacer que tu texto sea hermoso. El próximo paso para convertirse en un maestro de estilo es aprender a controlar el tamaño de los componentes .

Altura y ancho

La altura y el ancho de un componente determinan su tamaño en la pantalla.

Dimensiones fijas

La forma más sencilla de establecer las dimensiones de un componente es agregar un estilo fijo `width`y `height`un estilo. Todas las dimensiones en React Native no tienen unidades y representan píxeles independientes de la densidad.

```
import React, { Component } from 'react';
import { AppRegistry, View } from 'react-native';

export default class FixedDimensionsBasics extends Component {
  render() {
    return (
      <View>
        <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'skyblue'}} />
        <View style={{width: 150, height: 150, backgroundColor: 'steelblue'}} />
      </View>
    );
  }
}
```

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => FixedDimensionsBasics);

Establecer las dimensiones de esta manera es común para los componentes que siempre deben representar exactamente el mismo tamaño, independientemente de las dimensiones de la pantalla.

Dimensiones flexibles

Úselo `flex` en el estilo de un componente para que el componente se expanda y reduzca de forma dinámica en función del espacio disponible. Normalmente usará `flex: 1`, que le dice a un componente que complete todo el espacio disponible, compartido de manera equitativa entre cada componente con el mismo padre. Cuanto mayor sea el `flex` dado, mayor será la proporción de espacio que tomará un componente en comparación con sus hermanos.

> Un componente solo puede expandirse para llenar el espacio disponible si su padre tiene dimensiones mayores que 0. Si un padre no tiene ni un fijo `width` y `height` ni `flex`, el padre tendrá dimensiones de 0 y los `flex` niños no serán visibles.

```
import React, { Component } from 'react';
import { AppRegistry, View } from 'react-native';

export default class FlexDimensionsBasics extends Component {
  render() {
    return (
      // Try removing the `flex: 1` on the parent View.
```

```
      // The parent will not have dimensions, so the children can't expand.
      // What if you add `height: 300` instead of `flex: 1`?
      <View style={{flex: 1}}>
        <View style={{flex: 1, backgroundColor: 'powderblue'}} />
        <View style={{flex: 2, backgroundColor: 'skyblue'}} />
        <View style={{flex: 3, backgroundColor: 'steelblue'}} />
      </View>
    );
  }
}

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => FlexDimensionsBasics);
```

Después de que pueda controlar el tamaño de un componente, el próximo paso es aprender a diseñarlo en la pantalla .

Diseño con Flexbox

Un componente puede especificar el diseño de sus hijos utilizando el algoritmo FlexBox. Flexbox está diseñado para proporcionar un diseño consistente en diferentes tamaños de pantalla.

Es normal que se utilice una combinación de `flexDirection`, `alignItems`y `justifyContent` para lograr la disposición correcta.

> Flexbox funciona de la misma manera en React Native que en CSS en la web, con algunas excepciones. Los valores predeterminados son diferentes, con el valor predeterminado `flexDirection` en `column` lugar de `row`, y el `flex` parámetro solo admite un solo número.

Dirección Flex

Agregar `flexDirection` a un componente `style` determina el **eje principal** de su diseño. ¿Los niños deben organizarse horizontalmente (`row`) o verticalmente (`column`)? El valor por defecto es `column`.

```
import React, { Component } from 'react';
import { AppRegistry, View } from 'react-native';

export default class FlexDirectionBasics extends Component {
  render() {
    return (
      // Try setting `flexDirection` to `column`.
      <View style={{flex: 1, flexDirection: 'row'}}>
        <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
        <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />
        <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />
      </View>
    );
  }
};

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => FlexDirectionBasics);
```

Justificar contenido

Agregar `justifyContent` al estilo de un componente determina la **distribución**de los niños a lo largo del **eje primario** . ¿Deben los niños ser distribuidos al principio, al centro, al final o espaciados uniformemente? Las opciones disponibles son `flex-start`, `center`, `flex-end`, `space-around`, y `space-between`.

```
import React, { Component } from 'react';
import { AppRegistry, View } from 'react-native';

export default class JustifyContentBasics extends Component {
  render() {
    return (
```

```
      // Try setting `justifyContent` to `center`.
      // Try setting `flexDirection` to `row`.
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'space-between',
      }}>
        <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
        <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />
        <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />
      </View>
    );
  }
};

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => JustifyContentBasics);
```

Alinear elementos

Agregar `alignItems` al estilo de un componente determina la **alineación** de los niños a lo largo del **eje secundario** (si el eje primario es `row`, entonces el secundario es `column`, y viceversa). ¿Los niños deben estar alineados al principio, al centro, al final o estirados para llenar? Las opciones disponibles son `flex-start`, `center`, `flex-end`, y `stretch`.

> Para `stretch` tener un efecto, los niños no deben tener una dimensión fija a lo largo del eje secundario. En el siguiente ejemplo, la configuración `alignItems: stretch` no hace nada hasta que `width: 50` se elimine de los elementos secundarios.

```
import React, { Component } from 'react';
import { AppRegistry, View } from 'react-native';

export default class AlignItemsBasics extends Component {
  render() {
    return (
      // Try setting `alignItems` to 'flex-start'
      // Try setting `justifyContent` to `flex-end`.
      // Try setting `flexDirection` to `row`.
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'center',
      }}>
        <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
        <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />
        <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />
      </View>
    );
```

```
  }
};

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => AlignItemsBasics);
```

Yendo más profundo

Hemos cubierto los conceptos básicos, pero hay muchos otros estilos que puede necesitar para los diseños. La lista completa de accesorios que controlan el diseño está documentada aquí .

Nos estamos acercando a poder construir una aplicación real. Una cosa que aún nos falta es una forma de tomar las aportaciones del usuario, así que pasemos a aprender a manejar el ingreso de texto con el componente TextInput .

Manejo de entrada de texto

`TextInput`es un componente básico que permite al usuario ingresar texto. Tiene un `onChangeText` apoyo que toma una función para ser llamada cada vez que el texto cambió, y un `onSubmitEditing` apoyo que toma una función para llamar cuando se envía el texto.

Por ejemplo, digamos que a medida que el usuario escribe, está traduciendo sus palabras a un idioma diferente. En este nuevo idioma, cada palabra se escribe de la misma manera: . Entonces la oración "Hola, Bob" se traduciría como "".

```
import React, { Component } from 'react';
import { AppRegistry, Text, TextInput, View } from 'react-native';

export default class PizzaTranslator extends Component {
  constructor(props) {
    super(props);
    this.state = {text: ''};
  }

  render() {
    return (
      <View style={{padding: 10}}>
        <TextInput
          style={{height: 40}}
          placeholder="Type here to translate!"
          onChangeText={(text) => this.setState({text})}
        />
        <Text style={{padding: 10, fontSize: 42}}>
          {this.state.text.split(' ').map((word) => word && '').join(' ')}
        </Text>
      </View>
    );
  }
}

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => PizzaTranslator);
```

En este ejemplo, almacenamos `text`en el estado, porque cambia con el tiempo.

Hay muchas más cosas que podría querer hacer con una entrada de texto. Por ejemplo, puede validar el texto dentro mientras el usuario escribe. Para obtener ejemplos más detallados, consulte los documentos React sobre componentes controlados o los documentos de referencia para TextInput .

La entrada de texto es una de las formas en que el usuario interactúa con la aplicación. A continuación, veamos otro tipo de entrada y aprenderemos a manejar los toques .

Manejo de toques

Los usuarios interactúan con aplicaciones móviles principalmente a través del tacto. Pueden usar una combinación de gestos, como tocar un botón, desplazarse por una lista o hacer zoom en un mapa. React Native proporciona componentes para manejar todo tipo de gestos comunes, así como un sistema completo de respuesta de gestos para permitir un reconocimiento de gestos más avanzado, pero el único componente que probablemente le interese es el botón básico.

Mostrando un botón básico

Button proporciona un componente de botón básico que se procesa muy bien en todas las plataformas. El ejemplo mínimo para mostrar un botón se ve así:

```
<Button
        onPress={() => { Alert.alert('You tapped the button!')}}
        title="Press Me"
/>
```

Esto representará una etiqueta azul en iOS, y un rectángulo azul redondeado con texto blanco en Android. Al presionar el botón se llamará a la función "onPress", que en este caso muestra una alerta emergente. Si lo desea, puede especificar un accesorio de "color" para cambiar el color de su botón.

Siga adelante y juegue con el `Button` componente usando el ejemplo a continuación. Puede seleccionar en qué plataforma se realiza una vista previa de su aplicación haciendo clic en la barra de la esquina inferior derecha y luego hacer clic en "Tocar para reproducir" para obtener una vista previa de la aplicación.

```
import React, { Component } from 'react';
import { Alert, AppRegistry, Button, StyleSheet, View } from 'react-native';

export default class ButtonBasics extends Component {
  _onPressButton() {
    Alert.alert('You tapped the button!')
  }

  render() {
    return (
      <View style={styles.container}>
        <View style={styles.buttonContainer}>
          <Button
            onPress={this._onPressButton}
            title="Press Me"
          />
        </View>
```

```
        <View style={styles.buttonContainer}>
          <Button
            onPress={this._onPressButton}
            title="Press Me"
            color="#841584"
          />
        </View>
        <View style={styles.alternativeLayoutButtonContainer}>
          <Button
            onPress={this._onPressButton}
            title="This looks great!"
          />
          <Button
            onPress={this._onPressButton}
            title="OK!"
            color="#841584"
          />
        </View>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
   flex: 1,
   justifyContent: 'center',
  },
  buttonContainer: {
    margin: 20
  },
  alternativeLayoutButtonContainer: {
    margin: 20,
    flexDirection: 'row',
    justifyContent: 'space-between'
  }
})

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => ButtonBasics);
```

Tocables

Si el botón básico no se ve bien para su aplicación, puede crear su propio botón utilizando cualquiera de los componentes "Touchable" proporcionados por React Native. Los componentes "Touchable" brindan la capacidad de capturar gestos de tapping, y pueden mostrar comentarios cuando se reconoce un gesto. Sin embargo, estos

componentes no proporcionan ningún estilo predeterminado, por lo que deberá hacer un poco de trabajo para que se vean bien en su aplicación.

El componente "táctil" que use dependerá del tipo de comentarios que quiera proporcionar:

- En general, puede usar **TouchableHighlight en** cualquier lugar donde use un botón o un enlace en la web. El fondo de la vista se oscurecerá cuando el usuario presione el botón.
- Puede considerar utilizar **TouchableNativeFeedback** en Android para visualizar las ondas de reacción de la superficie de la tinta que responden al tacto del usuario.
- **TouchableOpacity** se puede utilizar para proporcionar retroalimentación al reducir la opacidad del botón, permitiendo que se vea el fondo mientras el usuario presiona hacia abajo.
- Si necesita manejar un gesto de toque, pero no desea que se muestre ningún comentario, use **Touchable sin** retroalimentación .

En algunos casos, es posible que desee detectar cuándo un usuario presiona y mantiene una vista durante un período de tiempo determinado. Estas pulsaciones largas se pueden manejar pasando una función a los `onLongPress` puntales de cualquiera de los componentes "Touchable".

Veamos todo esto en acción:

```
import React, { Component } from 'react';
import { Alert, AppRegistry, Platform, StyleSheet, Text, TouchableHighlight,
TouchableOpacity, TouchableNativeFeedback, TouchableWithoutFeedback, View } from 'react-
native';

export default class Touchables extends Component {
  _onPressButton() {
    Alert.alert('You tapped the button!')
  }

  _onLongPressButton() {
    Alert.alert('You long-pressed the button!')
  }


  render() {
    return (
      <View style={styles.container}>
        <TouchableHighlight onPress={this._onPressButton} underlayColor="white">
          <View style={styles.button}>
            <Text style={styles.buttonText}>TouchableHighlight</Text>
          </View>
        </TouchableHighlight>
        <TouchableOpacity onPress={this._onPressButton}>
          <View style={styles.button}>
            <Text style={styles.buttonText}>TouchableOpacity</Text>
          </View>
        </TouchableOpacity>
        <TouchableNativeFeedback
```

```
            onPress={this._onPressButton}
            background={Platform.OS === 'android' ?
TouchableNativeFeedback.SelectableBackground() : ''}>
          <View style={styles.button}>
            <Text style={styles.buttonText}>TouchableNativeFeedback</Text>
          </View>
        </TouchableNativeFeedback>
        <TouchableWithoutFeedback
            onPress={this._onPressButton}
            >
          <View style={styles.button}>
            <Text style={styles.buttonText}>TouchableWithoutFeedback</Text>
          </View>
        </TouchableWithoutFeedback>
        <TouchableHighlight onPress={this._onPressButton}
onLongPress={this._onLongPressButton} underlayColor="white">
          <View style={styles.button}>
            <Text style={styles.buttonText}>Touchable with Long Press</Text>
          </View>
        </TouchableHighlight>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    paddingTop: 60,
    alignItems: 'center'
  },
  button: {
    marginBottom: 30,
    width: 260,
    alignItems: 'center',
    backgroundColor: '#2196F3'
  },
  buttonText: {
    padding: 20,
    color: 'white'
  }
})

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => Touchables);
```

Listas de desplazamiento, deslizando páginas, y pellizcar para hacer zoom

Otro gesto comúnmente utilizado en aplicaciones móviles es el deslizamiento o la panorámica. Este gesto permite al usuario desplazarse por una lista de elementos o deslizar páginas por el contenido. Para manejar estos y otros gestos, aprenderemos cómo usar un ScrollView a continuación.

Usando un ScrollView

El ScrollView es un contenedor de desplazamiento genérico que puede alojar múltiples componentes y puntos de vista. Los elementos desplazables no necesitan ser homogéneos, y puede desplazarse tanto vertical como horizontalmente (estableciendo la `horizontal` propiedad).

Este ejemplo crea una vertical `ScrollView` con imágenes y texto mezclados.

```
import React, { Component } from 'react';
import { AppRegistry, ScrollView, Image, Text } from 'react-native';

export default class IScrolledDownAndWhatHappenedNextShockedMe extends Component {
  render() {
      return (
        <ScrollView>
          <Text style={{fontSize:96}}>Scroll me plz</Text>
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Text style={{fontSize:96}}>If you like</Text>
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Text style={{fontSize:96}}>Scrolling down</Text>
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Text style={{fontSize:96}}>What's the best</Text>
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Text style={{fontSize:96}}>Framework around?</Text>
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Text style={{fontSize:80}}>React Native</Text>
```

```
        </ScrollView>
    );
  }
}

// skip these lines if using Create React Native App
AppRegistry.registerComponent(
  'AwesomeProject',
  () => IScrolledDownAndWhatHappenedNextShockedMe);
```

ScrollViews se puede configurar para permitir la paginación a través de vistas utilizando gestos de deslizamiento mediante el uso de los `pagingEnabled` accesorios. Deslizar horizontalmente entre vistas también se puede implementar en Android utilizando el componente ViewPagerAndroid .

Un ScrollView con un solo elemento se puede utilizar para permitir al usuario ampliar el contenido. Configurar las `maximumZoomScale` y `minimumZoomScale` los apoyos y el usuario será capaz de utilizar y ampliar pizca gestos para acercar y alejar.

ScrollView funciona mejor para presentar una pequeña cantidad de cosas de un tamaño limitado. Todos los elementos y vistas de a `ScrollView`se representan, incluso si no se muestran actualmente en la pantalla. Si tiene una larga lista de más elementos que pueden caber en la pantalla, debería usar un `FlatList` en su lugar. Así que aprendamos sobre las vistas de lista a continuación.

Usar vistas de lista

React Native proporciona un conjunto de componentes para presentar listas de datos. Generalmente, querrá usar FlatList o SectionList .

El `FlatList` componente muestra una lista de desplazamiento de datos cambiantes, pero estructurados de manera similar. `FlatList` funciona bien para largas listas de datos, donde la cantidad de elementos puede cambiar con el tiempo. A diferencia de lo más genérico `ScrollView`, el `FlatList` único representa elementos que se muestran actualmente en la pantalla, no todos los elementos a la vez.

El `FlatList` componente requiere dos apoyos: `data` y `renderItem`. `Data` es la fuente de información para la lista. `RenderItem` toma un elemento de la fuente y devuelve un componente formateado para representar.

Este ejemplo crea un simple `FlatList` de datos codificados. Cada elemento en los `data` accesorios se representa como un `Text` componente. El `FlatListBasics` componente luego representa el `FlatList` y todos los `Text` componentes.

```
import React, { Component } from 'react';
import { AppRegistry, SectionList, StyleSheet, Text, View } from 'react-native';

export default class SectionListBasics extends Component {
  render() {
    return (
      <View style={styles.container}>
        <SectionList
          sections={[
            {title: 'D', data: ['Devin']},
            {title: 'J', data: ['Jackson', 'James', 'Jillian', 'Jimmy', 'Joel', 'John',
'Julie']},
          ]}
          renderItem={({item}) => <Text style={styles.item}>{item}</Text>}
          renderSectionHeader={({section}) => <Text
style={styles.sectionHeader}>{section.title}</Text>}
        />
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
   flex: 1,
   paddingTop: 22
  },
  sectionHeader: {
    paddingTop: 2,
    paddingLeft: 10,
```

```
    paddingRight: 10,
    paddingBottom: 2,
    fontSize: 14,
    fontWeight: 'bold',
    backgroundColor: 'rgba(247,247,247,1.0)',
  },
  item: {
    padding: 10,
    fontSize: 18,
    height: 44,
  },
})

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => SectionListBasics);
```

Si desea convertir un conjunto de datos dividido en secciones lógicas, tal vez con encabezados de sección, similar a `UITableView`s en iOS, entonces SectionListes el camino a seguir.

```
import React, { Component } from 'react';
import { AppRegistry, SectionList, StyleSheet, Text, View } from 'react-native';

export default class SectionListBasics extends Component {
  render() {
    return (
      <View style={styles.container}>
        <SectionList
          sections={[
            {title: 'D', data: ['Devin']},
            {title: 'J', data: ['Jackson', 'James', 'Jillian', 'Jimmy', 'Joel', 'John',
'Julie']},
          ]}
          renderItem={({item}) => <Text style={styles.item}>{item}</Text>}
          renderSectionHeader={({section}) => <Text
style={styles.sectionHeader}>{section.title}</Text>}
        />
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
   flex: 1,
   paddingTop: 22
  },
  sectionHeader: {
```

```
    paddingTop: 2,
    paddingLeft: 10,
    paddingRight: 10,
    paddingBottom: 2,
    fontSize: 14,
    fontWeight: 'bold',
    backgroundColor: 'rgba(247,247,247,1.0)',
  },
  item: {
    padding: 10,
    fontSize: 18,
    height: 44,
  },
})

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => SectionListBasics);
```

Uno de los usos más comunes para una vista de lista es mostrar los datos que obtiene de un servidor. Para hacer eso, necesitarás aprender sobre la creación de redes en React Native .

Redes

Muchas aplicaciones móviles necesitan cargar recursos desde una URL remota. Es posible que desee realizar una solicitud POST a una API REST, o simplemente necesita obtener un fragmento de contenido estático de otro servidor.

Utilizando Fetch

React Native proporciona la API de obtención para sus necesidades de red. La búsqueda te resultará familiar si has utilizado `XMLHttpRequest` u otras API de red antes. Puede consultar la guía de MDN sobre el uso de Fetch para obtener información adicional.

Hacer solicitudes

Para obtener contenido de una URL arbitraria, simplemente pase la URL a buscar:

```
fetch('https://mywebsite.com/mydata.json')
```

Fetch también toma un segundo argumento opcional que le permite personalizar la solicitud HTTP. Es posible que desee especificar encabezados adicionales o realizar una solicitud POST:

```
fetch('https://mywebsite.com/endpoint/', {
        method: 'POST', headers: {
                'Accept': 'application/json',
                'Content-Type': 'application/json',
        },
        body: JSON.stringify({
                firstParam: 'yourValue',
                secondParam: 'yourOtherValue',
        })
})
```

Eche un vistazo a los documentos de solicitud de obtención para obtener una lista completa de propiedades.

Manejando la respuesta

Los ejemplos anteriores muestran cómo puedes hacer una solicitud. En muchos casos, querrás hacer algo con la respuesta.

La conexión en red es una operación inherentemente asíncrona. Los métodos de búsqueda devolverán una Promesa que hace que sea sencillo escribir código que funcione de manera asíncrona:

```
function getMoviesFromApiAsync() {
        return fetch('https://facebook.github.io/react-native/movies.json')
                .then((response) => response.json())
                .then((responseJson) => {
                        return responseJson.movies;
                })
                .catch((error) => {
                        console.error(error);
                });
}
```

También puede usar la sintaxis propuesta ES2017 `async`/`await` en una aplicación React Native:

```
async function getMoviesFromApi() {
        try {
                let response = await fetch('https://facebook.github.io/react-native/movies.json');
                let responseJson = await response.json();
                return responseJson.movies;
        } catch(error) {
                console.error(error);
        }
}
```

No se olvide de detectar los errores que puedan surgir `fetch`, de lo contrario, se eliminarán en silencio.

```
import React, { Component } from 'react';
import { ActivityIndicator, ListView, Text, View } from 'react-native';

export default class Movies extends Component {
  constructor(props) {
    super(props);
    this.state = {
      isLoading: true
    }
  }

  componentDidMount() {
    return fetch('https://facebook.github.io/react-native/movies.json')
      .then((response) => response.json())
      .then((responseJson) => {
        let ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});
        this.setState({
          isLoading: false,
          dataSource: ds.cloneWithRows(responseJson.movies),
        }, function() {
          // do something with new state
        });
      })
      .catch((error) => {
        console.error(error);
      });
  }

  render() {
    if (this.state.isLoading) {
      return (
        <View style={{flex: 1, paddingTop: 20}}>
          <ActivityIndicator />
        </View>
      );
```

```
    }

    return (
      <View style={{flex: 1, paddingTop: 20}}>
        <ListView
          dataSource={this.state.dataSource}
          renderRow={(rowData) => <Text>{rowData.title}, {rowData.releaseYear}</Text>}
        />
      </View>
    );
  }
}
```

> De manera predeterminada, iOS bloqueará cualquier solicitud que no esté encriptada mediante SSL. Si necesita recuperar desde una URL sin formato (una que comience con `http`), primero deberá agregar una excepción de Seguridad de transporte de aplicaciones. Si sabe de antemano a qué dominios necesitará acceder, es más seguro agregar excepciones solo para esos dominios; si los dominios no se conocen hasta el tiempo de ejecución, puede desactivar completamente ATS . Sin embargo, tenga en cuenta que a partir de enero de 2017, la revisión de la App Store de Apple requerirá una justificación razonable para deshabilitar ATS . Vea la documentación de Apple para más información.

Uso de otras bibliotecas de red

La API XMLHttpRequest está integrada en React Native. Esto significa que puede usar bibliotecas de terceros como frisbee o axios que dependen de ella, o puede usar la API XMLHttpRequest directamente si lo prefiere.

```
var request = new XMLHttpRequest();
request.onreadystatechange = (e) => {
        if (request.readyState !== 4) {
                return;
        }
        if (request.status === 200) {
                console.log('success', request.responseText);
        } else {
                console.warn('error');
        }
};
request.open('GET', 'https://mywebsite.com/endpoint/');
request.send();
```

> El modelo de seguridad para XMLHttpRequest es diferente de la web, ya que no existe el concepto de CORS en las aplicaciones nativas.

Soporte de WebSocket

React Native también es compatible con WebSockets , un protocolo que proporciona canales de comunicación full-duplex a través de una única conexión TCP.

```
var ws = new WebSocket('ws://host.com/path');
ws.onopen = () => {
        // connection opened
        ws.send('something');
        // send a message
};
ws.onmessage = (e) => {
        // a message was received
        console.log(e.data);
};
```

```
ws.onerror = (e) => {
        // an error occurred
        console.log(e.message);
};
ws.onclose = (e) => {
        // connection closed
        console.log(e.code, e.reason);
};
```

¡Choca esos cinco!

Si has llegado aquí leyendo linealmente a través del tutorial, entonces eres un ser humano bastante impresionante. Felicitaciones. A continuación, es posible que desee comprobar todas las cosas interesantes que la comunidad hace con React Native .

Más recursos

Si acabas de leer este sitio web, deberías poder construir una aplicación React Native muy buena. Pero React Native no es solo un producto hecho por una compañía, es una comunidad de miles de desarrolladores. Entonces, si estás interesado en React Native, aquí hay algunas cosas relacionadas que tal vez quieras consultar.

Bibliotecas Populares

Si usa React Native, probablemente ya sepa sobre React . Así que me siento un poco tonto al mencionar esto. Pero si no lo has hecho, revisa Reaccionar: es la mejor manera de crear un sitio web moderno.

Una pregunta común es cómo manejar el "estado" de su aplicación React Native. La biblioteca más popular para esto es Redux . No temas la frecuencia con la que Redux usa la palabra "reducer": es una biblioteca bastante simple, y también hay una buena serie de videos que la explican.

Si estás buscando una biblioteca que haga algo específico, echa un vistazo a Awesome React Native , una lista de componentes curada que también tiene demostraciones, artículos y más.

Ejemplos

¡Prueba aplicaciones de Showcase para ver lo que React Native es capaz de hacer! También hay algunas aplicaciones de ejemplo en GitHub . Puede ejecutar las aplicaciones en un simulador o dispositivo, y puede ver el código fuente de estas aplicaciones, que es perfecto.

Las personas que crearon la aplicación para la conferencia F8 de Facebook en 2016 también abrieron el código y escribieron una serie detallada de tutoriales . Esto es útil si desea un ejemplo más profundo que sea más realista que la mayoría de las aplicaciones de muestra que existen.

Prolongación de React Native

- Buscando un componente? JS.coach
- Otros desarrolladores escriben y publican módulos React Native para npm y los abren en GitHub.
- Hacer módulos ayuda a crecer el ecosistema y la comunidad de React Native. Recomendamos escribir módulos para sus casos de uso y compartirlos en npm.
- Lea las guías sobre módulos nativos (iOS , Android) y componentes nativos de interfaz de usuario (iOS , Android) si está interesado en ampliar la funcionalidad nativa.

Herramientas de desarrollo

Nuclide es el IDE que Facebook usa internamente para el desarrollo de JavaScript. La característica principal de Nuclide es su capacidad de depuración. También tiene un gran soporte de flujo en línea. VS Code es otro IDE que es popular entre los desarrolladores de JavaScript.

Ignite es un kit de inicio que usa Redux y algunas bibliotecas de IU comunes. Tiene una CLI para generar aplicaciones, componentes y contenedores. Si te gustan todas las opciones tecnológicas individuales, Ignite podría ser perfecto para ti.

CodePush es un servicio de Microsoft que facilita la implementación de actualizaciones en vivo en su aplicación React Native. Si no te gusta pasar por el proceso de la tienda de aplicaciones para implementar pequeños ajustes, y tampoco te gusta configurar tu propio backend, prueba CodePush.

Expo es una aplicación de entorno de desarrollo más que se centra en permitirle crear aplicaciones React Native en el entorno de desarrollo de la Expo, sin tocar Xcode o Android Studio. Si desea que React Native sea aún más JavaScripty y webby, consulte Expo.
Las herramientas para desarrolladores de React son excelentes para la depuración de aplicaciones nativas Reaccionar y Reaccionar.

Donde reaccionan los nativos Hang Out

El grupo de Facebook de React Native Community tiene miles de desarrolladores, y es bastante activo. Ven allí para mostrar tu proyecto o pregunta cómo otras personas resolvieron problemas similares.
Reactiflux es un chat de Discordia donde se produce una gran cantidad de discusiones relacionadas con React, incluido React Native. Discord es como Slack, excepto que funciona mejor para proyectos de código abierto con un billón de contribuyentes. Mira el canal # reaction-native.
La cuenta de React Twitter cubre tanto React como React Native. Siga la cuenta y el blog de React Native Twitter para averiguar qué está sucediendo en el mundo de React Native.
Hay muchos Meetups Native React que suceden en todo el mundo. A menudo, también hay contenido React Native en las reuniones de React.
A veces tenemos conferencias de React. Publicamos los videos de React.js Conf 2017 y React.js Conf 2016 , y probablemente tendremos más conferencias en el futuro también. Manténganse al tanto. También puede encontrar una lista de conferencias dedicadas a React Native aquí .

Guías

Componentes y API

React Native proporciona una serie de componentes integrados. Encontrará una lista completa de componentes y API en la barra lateral a la izquierda. Si no está seguro de dónde comenzar, eche un vistazo a las siguientes categorías:

- Componentes básicos
- Interfaz de usuario
- Vistas de Listas
- iOS-específico
- Android específico
- Otros

No está limitado a los componentes y API incluidos con React Native. React Native es una comunidad de miles de desarrolladores. Si busca una biblioteca que haga algo específico, busque en el registro npm los paquetes que mencionen react-native , o consulte Awesome React Native para obtener una lista curada.

Componentes básicos

La mayoría de las aplicaciones terminarán usando uno de estos componentes básicos. Deseará familiarizarse con todo esto si es nuevo en React Native.

Vista
El componente más fundamental para construir una interfaz de usuario.

Texto
Un componente para mostrar texto.

Imagen
Un componente para mostrar imágenes.

Entrada de texto
Un componente para ingresar texto en la aplicación a través de un teclado.

ScrollView
Proporciona un contenedor desplazable que puede albergar múltiples componentes y vistas.

StyleSheet
Proporciona una capa de abstracción similar a las hojas de estilo CSS.

Interfaz de usuario

Renderice controles comunes de la interfaz de usuario en cualquier plataforma usando los siguientes componentes. Para componentes específicos de plataforma, sigue leyendo.

Botón

Un componente de botón básico para manejar toques que deberían reproducirse en cualquier plataforma.

Picker

Renderiza el componente selector nativo en iOS y Android.

Control deslizante

Un componente utilizado para seleccionar un valor único de un rango de valores.

Switch

Renderiza una entrada booleana.

Vista de lista

A diferencia de los más genéricos `ScrollView`, los siguientes componentes de vista de lista solo muestran los elementos que se muestran actualmente en la pantalla. Esto los convierte en una gran opción para mostrar largas listas de datos.

FlatList

Un componente para renderizar listas desplegables de rendimiento.

SectionList

Me gusta `FlatList`, pero para listas seccionadas.

iOS Components and APIs

Muchos de los siguientes componentes proporcionan envoltorios para las clases de UIKit de uso común.

ActionSheetIOS

API para mostrar una hoja de acción de iOS o una hoja de compartir.

AlertIOS

Cree un cuadro de diálogo de alerta de iOS con un mensaje o cree un aviso para la entrada del usuario.

DatePickerIOS

Representa un selector de fecha / hora (selector) en iOS.

ImagePickerIOS

Renderiza un selector de imágenes en iOS.

NavigatorIOS

Un contenedor `UINavigationController`, lo que le permite implementar una pila de navegación.

ProgressViewIOS

Renderiza un `UIProgressView` iOS.

PushNotificationIOS

Maneje las notificaciones automáticas para su aplicación, incluido el manejo de permisos y el número de la insignia del icono.

SegmentedControlIOS

Renderiza un `UISegmentedControl` iOS.

TabBarIOS

Renderiza un `UITabViewController` iOS. Usar con TabBarIOS.Item .

Componentes y API de Android

Muchos de los siguientes componentes proporcionan envoltorios para las clases de Android más utilizadas.

BackHandler

Detectar pulsaciones de botón de hardware para navegación hacia atrás.

DatePickerAndroid

Abre el diálogo selector de fecha estándar de Android.

DrawerLayoutAndroid

Renders a `DrawerLayout` en Android.

PermisosAndroid

Proporciona acceso al modelo de permisos introducido en Android M.

ProgressBarAndroid

Renders a `ProgressBar` en Android.

TimePickerAndroid

Abre el cuadro de diálogo estándar del selector de tiempo de Android.

ToastAndroid

Crea una alerta de Android Toast.

ToolbarAndroid

Renders a `Toolbar` en Android.

ViewPagerAndroid

Contenedor que permite virar a la izquierda y a la derecha entre vistas de niños.

Otros

Estos componentes pueden ser útiles para ciertas aplicaciones. Para obtener una lista exhaustiva de componentes y API, consulte la barra lateral a la izquierda.

ActivityIndicator

Muestra un indicador de carga circular.

Alert

Inicia un diálogo de alerta con el título y el mensaje especificados.

Animated

Una biblioteca para crear animaciones fluidas y potentes que son fáciles de construir y mantener.

CameraRoll

Brinda acceso al rollo / galería de la cámara local.

Portapapeles

Proporciona una interfaz para configurar y obtener contenido desde el portapapeles en iOS y Android.

Dimensiones

Proporciona una interfaz para obtener las dimensiones del dispositivo.

KeyboardAvoidingView

Proporciona una vista que se mueve fuera del camino del teclado virtual automáticamente.

Enlace

Proporciona una interfaz general para interactuar con enlaces de aplicaciones entrantes y salientes.

Modal

Proporciona una forma simple de presentar el contenido por encima de una vista adjunta.

PixelRatio

Proporciona acceso a la densidad de píxeles del dispositivo.

RefreshControl

Este componente se usa dentro de a `ScrollView` para agregar funcionalidad pull to refresh.

Barra de estado

Componente para controlar la barra de estado de la aplicación.

WebView

Un componente que representa el contenido web en una vista nativa.

Código específico de la plataforma

Al crear una aplicación multiplataforma, querrá volver a utilizar la mayor cantidad de código posible. Pueden surgir situaciones en las que tenga sentido que el código sea diferente; por ejemplo, puede que desee implementar componentes visuales separados para iOS y Android.

React Native proporciona dos formas de organizar fácilmente su código y separarlo por plataforma:

- Usando el `Platform` módulo .
- Usar extensiones de archivos específicas de la plataforma .

Ciertos componentes pueden tener propiedades que funcionan solo en una plataforma. Todos estos accesorios están anotados `@platform`y tienen una pequeña insignia al lado de ellos en el sitio web.

Módulo de plataforma

React Native proporciona un módulo que detecta la plataforma en la que se ejecuta la aplicación. Puede usar la lógica de detección para implementar el código específico de la plataforma. Utilice esta opción cuando solo las partes pequeñas de un componente son específicas de la plataforma.

```
import { Platform, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
        height: (Platform.OS === 'ios') ? 200 : 100,
});
```

`Platform.OS`será `ios`cuando se ejecuta en iOS y `android`cuando se ejecuta en Android.

También hay un `Platform.select`método disponible, que dado un objeto que contiene Platform.OS como claves, devuelve el valor de la plataforma en la que se está ejecutando actualmente.

```
import { Platform, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
        container: {
                flex: 1,
                ...Platform.select({
                        ios: {
                                backgroundColor: 'red',
                        },
                        android: {
                                backgroundColor: 'blue',
                        },
                }),
        },
});
```

Esto dará como resultado un contenedor que tenga `flex: 1`ambas plataformas, un color de fondo rojo en iOS y un color de fondo azul en Android.

Como acepta `any` valor, también puede usarlo para devolver el componente específico de la plataforma, como se muestra a continuación:

```
const Component = Platform.select({
        ios: () => require('ComponentIOS'),
        android: () => require('ComponentAndroid'),
})();
```

```
<Component />;
```

Detectando la versión de Android

En Android, el `Platform` módulo también se puede usar para detectar la versión de la plataforma Android en la que se ejecuta la aplicación:

```
import { Platform } from 'react-native';

if (Platform.Version === 25) {
        console.log('Running on Nougat!');
}
```

Detectando la versión de iOS

En iOS, `Version` es el resultado de `-[UIDevice systemVersion]`, que es una cadena con la versión actual del sistema operativo. Un ejemplo de la versión del sistema es "10.3". Por ejemplo, para detectar el número de versión principal en iOS:

```
import { Platform } from 'react-native';

const majorVersionIOS = parseInt(Platform.Version, 10);
if (majorVersionIOS <= 9) {
        console.log('Work around a change in behavior');
}
```

Extensiones específicas de plataforma

Cuando el código específico de su plataforma es más complejo, debería considerar dividir el código en archivos separados. React Native detectará cuándo un archivo tiene una extensión `.ios.` o una `.android.` y cargará el archivo de plataforma relevante cuando sea necesario desde otros componentes.

Por ejemplo, supongamos que tiene los siguientes archivos en su proyecto:

```
BigButton.ios.js
BigButton.android.js
```

Luego puede requerir el componente de la siguiente manera:

```
const BigButton = require('./BigButton');
```

React Native recogerá automáticamente el archivo correcto según la plataforma en ejecución.

Navegando entre pantallas

Las aplicaciones móviles rara vez se componen de una sola pantalla. La gestión de la presentación de, y la transición entre, pantallas múltiples generalmente se maneja por lo que se conoce como un navegador.
Esta guía cubre los diversos componentes de navegación disponibles en React Native. Si recién está comenzando con la navegación, probablemente quiera usar Reaccionar Navegación . React Navigation ofrece una solución de navegación fácil de usar, con la capacidad de presentar una navegación de pila común y patrones de navegación con pestañas tanto en iOS como en Android. Como se trata de una implementación de JavaScript, proporciona la mayor cantidad de configurabilidad y flexibilidad cuando se integra con librerías de administración de estado como redux .
Si solo está apuntando a iOS, es posible que también desee comprobar NavigatorIOS como una forma de proporcionar una apariencia nativa con una configuración mínima, ya que proporciona una envoltura alrededor de la `UINavigationController` clase nativa . Sin embargo, este componente no funcionará en Android.
Si desea obtener un aspecto nativo en iOS y Android, o si está integrando React Native en una aplicación que ya administra la navegación de forma nativa, las siguientes bibliotecas proporcionan navegación nativa en ambas plataformas: navegación nativa , reacción navegación nativa .

Reaccionar Navegación

La solución comunitaria para la navegación es una biblioteca independiente que permite a los desarrolladores configurar las pantallas de una aplicación con solo unas pocas líneas de código.
El primer paso es instalarlo en su proyecto:

```
npm install --save react-navigation
```

Luego puede crear rápidamente una aplicación con una pantalla de inicio y una pantalla de perfil:

```
import { StackNavigator, } from 'react-navigation';

const App = StackNavigator({
        Home: { screen: HomeScreen },
        Profile: { screen: ProfileScreen },
});
```

Cada componente de pantalla puede establecer opciones de navegación, como el título del encabezado. Puede usar creadores de acciones en el `navigation`puntal para vincular a otras pantallas:

```
class HomeScreen extends React.Component {
        static navigationOptions = {
                title: 'Welcome',
        };
        render() {
                const { navigate } = this.props.navigation;
                return (
                        <Button
                                title="Go to Jane's profile"
                                onPress={() => navigate('Profile', { name: 'Jane' })
                                }
                        />
                );
```

```
        }
}
```

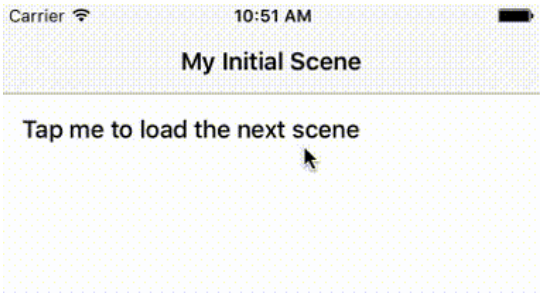
Los enrutadores de navegación React hacen que sea más fácil anular la lógica de navegación o integrarla en redux. Debido a que los enrutadores se pueden anidar uno dentro del otro, los desarrolladores pueden anular la lógica de navegación para un área de la aplicación sin hacer cambios generalizados.

Las vistas en React Navigation usan componentes nativos y la `Animated` biblioteca para entregar animaciones de 60 fps que se ejecutan en el hilo nativo. Además, las animaciones y los gestos se pueden personalizar fácilmente.

Para obtener una introducción completa a React Navigation, siga la Guía de inicio de React Navigation o busque otros documentos, como la introducción a Navigators .

NavigatorIOS

`NavigatorIOS` se ve y se siente como `UINavigationController`, porque en realidad está construido encima de él.




```
<NavigatorIOS
        initialRoute={{
                component: MyScene,
                title: 'My Initial Scene',
                passProps: { myProp: 'foo' },
        }}
/>
```

Al igual que otros sistemas de navegación, `NavigatorIOS` usa rutas para representar pantallas, con algunas diferencias importantes. El componente real que se representará se puede especificar utilizando la `component` clave en la ruta, y cualquier accesorio que se debe pasar a este componente se puede especificar en `passProps`. Un objeto "navegador" se pasa automáticamente como un apoyo al componente, lo que le permite llamar `push` y `pop`según sea necesario.

A medida que `NavigatorIOS` aprovecha la navegación UIKit nativa, se renderizará automáticamente una barra de navegación con un botón y título hacia atrás.

```
import React from 'react';
import PropTypes from 'prop-types';
import { Button, NavigatorIOS, Text, View } from 'react-native';

export default class NavigatorIOSApp extends React.Component {
        render() {
                return (
                        <NavigatorIOS
                                initialRoute={{
                                        component: MyScene,
                                        title: 'My Initial Scene',
                                        passProps: {index: 1},
                                }}
                                style={{flex: 1}}
                        />
                )
        }
}

class MyScene extends React.Component {
        static propTypes = {
                route: PropTypes.shape({
                        title: PropTypes.string.isRequired
                }),
                navigator: PropTypes.object.isRequired,
        }

        constructor(props, context) {
                super(props, context);
                this._onForward = this._onForward.bind(this);
        }

        onForward() {
                let nextIndex = ++this.props.index;
                this.props.navigator.push({
                        component: MyScene,
                        title: 'Scene ' + nextIndex,
                        passProps: {index: nextIndex}
                });
        }

        render() {
                return (
                        <View>
                                        <Text>Current Scene: { this.props.title }</Text>
                                <Button
                                        onPress={this._onForward}
                                        title="Tap me to load the next scene"
                                />
                        </View>
                )
        }
}
```

Consulte los `NavigatorIOS`documentos de referencia para obtener más información sobre este componente.

Imágenes

Recursos de imágenes estáticas

React Native proporciona una forma unificada de administrar imágenes y otros recursos multimedia en sus aplicaciones iOS y Android. Para agregar una imagen estática a su aplicación, colóquela en algún lugar de su árbol de código fuente y hágalo de la siguiente manera:

```
<Image source={require('./my-icon.png')} />
```

El nombre de la imagen se resuelve de la misma manera que se resuelven los módulos JS. En el ejemplo anterior, el empaquetador buscará `my-icon.png`en la misma carpeta que el componente que lo requiere. Además, si tiene `my-icon.ios.png` y `my-icon.android.png`, el empaquetador elegirá el archivo correcto para la plataforma.
También puede usar los sufijos `@2x`y `@3x`para proporcionar imágenes para diferentes densidades de pantalla. Si tiene la siguiente estructura de archivos:

```
. ├── button.js
  └── img
      ├── check@2x.png
      └── check@3x.png
```

... y el `button.js`código contiene:

```
<Image source={require('./img/check.png')} />
```

... el empaquetador empaquetará y publicará la imagen correspondiente a la densidad de la pantalla del dispositivo. Por ejemplo, `check@2x.png`se usará en un iPhone 7, mientras `check@3x.png`que se usará en un iPhone 7 Plus o un Nexus 5. Si no hay una imagen que coincida con la densidad de la pantalla, se seleccionará la mejor opción más cercana.
En Windows, es posible que deba reiniciar el paquete si agrega imágenes nuevas a su proyecto.
Aquí hay algunos beneficios que obtienes:

1. Mismo sistema en iOS y Android.
2. Las imágenes viven en la misma carpeta que su código JavaScript. Los componentes son autónomos.
3. No hay espacio de nombres global, es decir, no tiene que preocuparse por las colisiones de nombres.
4. Solo las imágenes que se usan realmente se empaquetarán en su aplicación.
5. Agregar y cambiar imágenes no requiere la recompilación de la aplicación, solo actualice el simulador como lo hace normalmente.
6. El empacador conoce las dimensiones de la imagen, no es necesario duplicarla en el código.
7. Las imágenes se pueden distribuir a través de paquetes npm .

Para que esto funcione, el nombre de la imagen `require` debe conocerse estáticamente.

```
// GOOD
<Image source={require('./my-icon.png')} />

// BAD
var icon = this.props.active ? 'my-icon-active' : 'my-icon-inactive';
<Image source={require('./' + icon + '.png')} />

// GOOD
var icon = this.props.active ? require('./my-icon-active.png') : require('./my-icon-inactive.png');
<Image source={icon} />
```

Tenga en cuenta que las fuentes de imagen requeridas de esta manera incluyen información de tamaño (ancho, alto) para la Imagen. Si necesita escalar la imagen dinámicamente (es decir, a través de flex), es posible que deba configurar manualmente `{ width: undefined, height: undefined }`el atributo de estilo.

Recursos estáticos sin imágenes

La `require` sintaxis descrita anteriormente también puede usarse para incluir de forma estática archivos de audio, video o documentos en su proyecto. La mayoría de los tipos de archivos más comunes se apoyan incluyendo `.mp3`, `.wav`, `.mp4`, `.mov`, `.html` y `.pdf`. Ver los valores predeterminados del paquete para la lista completa.

Puede agregar soporte para otros tipos creando un archivo de configuración del empaquetador (consulte el archivo de configuración del empaquetador para ver la lista completa de opciones de configuración).

Una advertencia es que los videos deben usar posicionamiento absoluto en lugar de `flexGrow`, ya que la información de tamaño no se pasa actualmente para los activos que no son imágenes. Esta limitación no ocurre para los videos que están vinculados directamente con Xcode o la carpeta de recursos para Android.

Imágenes de los recursos de la aplicación híbrida

Si está creando una aplicación híbrida (algunas UI en React Native, algunas UI en el código de plataforma), puede seguir usando imágenes que ya están incluidas en la aplicación.

Para las imágenes incluidas a través de los catálogos de activos de Xcode o en la carpeta dibujable de Android, use el nombre de la imagen sin la extensión:

```
<Image source={{uri: 'app_icon'}} style={{width: 40, height: 40}} />
```

Para imágenes en la carpeta de activos de Android, use el `asset:/`esquema:

```
<Image source={{uri: 'asset:/app_icon.png'}} style={{width: 40, height: 40}} />
```

Estos enfoques no proporcionan controles de seguridad. Depende de usted garantizar que esas imágenes estén disponibles en la aplicación. También debe especificar las dimensiones de la imagen manualmente.

Imágenes de red

Muchas de las imágenes que mostrará en su aplicación no estarán disponibles en tiempo de compilación, o querrá cargar algunas dinámicamente para mantener el tamaño binario abajo. A diferencia de los recursos estáticos, *deberá especificar manualmente las dimensiones de su imagen* . Es muy recomendable que use https también para satisfacer los requisitos de Seguridad de transporte de aplicaciones en iOS.

```
// GOOD
<Image source={{uri: 'https://facebook.github.io/react/img/logo_og.png'}} style={{width: 400, height: 400}} />

// BAD
<Image source={{uri: 'https://facebook.github.io/react/img/logo_og.png'}} />
```

Solicitudes de red para imágenes

Si desea establecer cosas tales como HTTP-Verb, Encabezados o un Cuerpo junto con la solicitud de imagen, puede hacer esto definiendo estas propiedades en el objeto fuente:

```
<Image source={{
                uri: 'https://facebook.github.io/react/img/logo_og.png',
                method: 'POST',
                headers: {
                        Pragma: 'no-cache'
                },
                body: 'Your Body goes here'
        }}

        style={{width: 400, height: 400}} />
```

Uri Data Images

En ocasiones, es posible que obtenga datos de imágenes codificadas de una llamada a la API REST. Puede usar el `'data:'` esquema uri para usar estas imágenes. Igual que para los recursos de red, *necesitará especificar manualmente las dimensiones de su imagen* .

> Esto se recomienda solo para imágenes muy pequeñas y dinámicas, como iconos en una lista de un DB.

```
// include at least width and height!
<Image style={{width: 51, height: 51, resizeMode: Image.resizeMode.contain}}
        source={{uri:
        '
SURBVGje7dSxCQBACARB+2/ab8WSYzYLYudDQYGBgYGBgYGBgYGBgZmcvDqYGBgmhivGQYGBgYGBgYGBgYGBgbmQw+P/eMrC5UTVAAA
AABJRU5ErkJggg==''}}/>
```

Control de caché (solo iOS)

En algunos casos, es posible que solo desee mostrar una imagen si ya está en la memoria caché local, es decir, un marcador de posición de baja resolución hasta que se encuentre disponible una resolución más alta. En otros casos, no importa si la imagen está desactualizada y si desea mostrar una imagen obsoleta para ahorrar ancho de banda.

La `cache` propiedad de origen le da control sobre cómo la capa de red interactúa con la memoria caché.

- `default`: Utilice la estrategia predeterminada de las plataformas nativas.
- `reload`: Los datos para la URL se cargarán desde la fuente de origen. No se deben usar datos de caché existentes para satisfacer una solicitud de carga de URL.
- `force-cache`: Los datos en caché existentes se usarán para satisfacer la solicitud, independientemente de su edad o fecha de vencimiento. Si no hay datos existentes en el caché que corresponden a la solicitud, los datos se cargan desde la fuente de origen.
- `only-if-cached`: Los datos de caché existentes se utilizarán para satisfacer una solicitud, independientemente de su edad o fecha de caducidad. Si no hay datos existentes en la memoria caché correspondientes a una solicitud de carga de URL, no se intenta cargar los datos desde la fuente de origen, y se considera que la carga ha fallado.

```
<Image source={{uri: 'https://facebook.github.io/react/img/logo_og.png', cache: 'only-if-cached'}}
        style={{width: 400, height: 400}} />
```

Imágenes locales del sistema de archivos

Consulte CameraRoll para ver un ejemplo del uso de recursos locales que están fuera de `Images.xcassets`.

Mejor cámara de rollo imagen

iOS guarda varios tamaños para la misma imagen en su Camera Roll, es muy importante seleccionar uno que esté lo más cerca posible por motivos de rendimiento. No querrá utilizar la imagen 3264x2448 de calidad completa como fuente cuando muestre una miniatura de 200x200. Si hay una coincidencia exacta, React Native la elegirá; de lo contrario, usará la primera que sea al menos un 50% más grande para evitar el desenfoque al cambiar el tamaño de un tamaño cercano. Todo esto se realiza de forma predeterminada, por lo que no tiene que preocuparse por escribir el código tedioso (y propenso a errores) para hacerlo usted mismo.

¿Por qué no se mide automáticamente todo?

En el navegador, si no le da un tamaño a una imagen, el navegador mostrará un elemento 0x0, descargará la imagen y luego la renderizará con el tamaño correcto. El gran problema con este comportamiento es que su interfaz de usuario saltará por todas partes a medida que se carguen las imágenes, lo que hace que la experiencia del usuario sea muy mala.

En React Native, este comportamiento no se implementa intencionalmente. Es más trabajo para el desarrollador conocer las dimensiones (o relación de aspecto) de la imagen remota de antemano, pero creemos que conduce a una mejor experiencia de usuario. Las imágenes estáticas cargadas desde el paquete de la aplicación a través de la `require('./my-icon.png')` sintaxis *se pueden dimensionar automáticamente* porque sus dimensiones están disponibles inmediatamente en el momento del montaje.

Por ejemplo, el resultado de `require('./my-icon.png')` podría ser:

```
{"__packager_asset":true,"uri":"my-icon.png","width":591,"height":573}
```

Fuente como un objeto

En React Native, una decisión interesante es que el `src` atributo se llama `source` y no toma una cadena sino un objeto con un `uri`atributo.

```
<Image source={{uri: 'something.jpg'}} />
```

En el lado de la infraestructura, la razón es que nos permite adjuntar metadatos a este objeto. Por ejemplo, si está usando `require('./my-icon.png')`, agregamos información sobre su ubicación y tamaño real (¡no confíe en este hecho, podría cambiar en el futuro!). Esto también es una prueba de futuro; por ejemplo, es posible que deseemos dar soporte a los sprites en algún momento, en lugar de generar `{uri: ...}`, podemos producir `{uri: ..., crop: {left: 10, top: 50, width: 20, height: 40}}`y respaldar de forma transparente el sprit en todos los sitios de llamadas existentes.

En el lado del usuario, esto le permite anotar el objeto con atributos útiles, como la dimensión de la imagen, para calcular el tamaño en el que se mostrará. Siéntase libre de usarlo como su estructura de datos para almacenar más información sobre su imagen .

Imagen de fondo a través del nido

Una solicitud de función común de desarrolladores familiarizados con la web es `background-image`. Para manejar este caso de uso, puede usar el `<ImageBackground>`componente, que tiene los mismos elementos que `<Image>`, y agregar los elementos sobre los que le gustaría superponerlos.

Es posible que no desee utilizar `<ImageBackground>`en algunos casos, ya que la implementación es muy simple. Consulte el código fuente de`<ImageBackground>` s para obtener más información y crear su propio componente personalizado cuando sea necesario.

```
Return (
        <ImageBackground source={...}>
                <Text>Inside</Text>
        </ImageBackground>
);
```

Estilos de radio de borde de iOS

Tenga en cuenta que el componente de imagen de iOS ignora actualmente las siguientes propiedades de estilo de radio de borde específicas de la esquina:

- `borderTopLeftRadius`
- `borderTopRightRadius`
- `borderBottomLeftRadius`
- `borderBottomRightRadius`

Decodificación fuera del hilo

La decodificación de imágenes puede tomar más de un marco de tiempo. Esta es una de las principales fuentes de caída de cuadros en la web porque la decodificación se realiza en el hilo principal. En React Native, la decodificación de imágenes se realiza en un hilo diferente. En la práctica, ya debe manejar el caso cuando la imagen aún no se haya descargado, por lo que mostrar el marcador de posición para algunos cuadros más mientras se está decodificando no requiere ningún cambio de código.

Animaciones

Las animaciones son muy importantes para crear una gran experiencia de usuario. Los objetos estacionarios deben vencer la inercia a medida que comienzan a moverse. Los objetos en movimiento tienen impulso y rara vez se detienen de inmediato. Las animaciones le permiten transmitir un movimiento físicamente creíble en su interfaz. React Native proporciona dos sistemas de animación complementarios:`Animated`para control granular e interactivo de valores específicos y `LayoutAnimation` para transacciones animadas de diseño global.

`Animated`API

La `Animated`API está diseñada para que sea muy fácil expresar de forma concisa una amplia variedad de patrones de animación e interacción interesantes de una manera muy eficiente. `Animated` se enfoca en relaciones declarativas entre entradas y salidas, con transformaciones configurables en el medio, y métodos simples `start`/`stop` para controlar la ejecución de la animación basada en el tiempo.

`Animated` exporta cuatro tipos de componentes animables: `View`, `Text`, `Image`, y `ScrollView`, pero también puede crear su propio utilizando `Animated.createAnimatedComponent()`.

Por ejemplo, una vista de contenedor que se desvanece cuando está montada puede verse así:

```
import React from 'react';
import { Animated, Text, View } from 'react-native';

class FadeInView extends React.Component {
  state = {
    fadeAnim: new Animated.Value(0),  // Initial value for opacity: 0
  }

  componentDidMount() {
    Animated.timing(                  // Animate over time
      this.state.fadeAnim,            // The animated value to drive
      {
        toValue: 1,                   // Animate to opacity: 1 (opaque)
        duration: 10000,              // Make it take a while
      }
    ).start();                        // Starts the animation
  }

  render() {
    let { fadeAnim } = this.state;

    return (
      <Animated.View                 // Special animatable View
        style={{
          ...this.props.style,
```

```
          opacity: fadeAnim,         // Bind opacity to animated value
        }}
      >
        {this.props.children}
      </Animated.View>
    );
  }
}

// You can then use your `FadeInView` in place of a `View` in your components:
export default class App extends React.Component {
  render() {
    return (
      <View style={{flex: 1, alignItems: 'center', justifyContent: 'center'}}>
        <FadeInView style={{width: 250, height: 50, backgroundColor: 'powderblue'}}>
          <Text style={{fontSize: 28, textAlign: 'center', margin: 10}}>Fading in</Text>
        </FadeInView>
      </View>
    )
  }
}
```

Analicemos lo que está pasando aquí. En el `FadeInView` constructor, se inicializa una nueva `Animated.Value` llamada `fadeAnim` como parte de `state`. La propiedad de opacidad en el `View` se asigna a este valor animado. Detrás de escena, el valor numérico se extrae y se usa para establecer la opacidad.
Cuando el componente se monta, la opacidad se establece en 0. Luego, se inicia una animación de aceleración en el `fadeAnim` valor animado, que actualizará todas sus asignaciones dependientes (en este caso, solo la opacidad) en cada cuadro a medida que el valor se anime a la valor final de 1.
Esto se hace de una manera optimizada que es más rápida que llamar `setState` y rehacer.
Como toda la configuración es declarativa, podremos implementar más optimizaciones que serialicen la configuración y ejecuten la animación en un hilo de alta prioridad.

Configurando animaciones

Las animaciones son muy configurables. Las funciones de aceleración personalizadas y predefinidas, los retrasos, las duraciones, los factores de disminución, las constantes de resorte y más se pueden ajustar según el tipo de animación.
`Animated` proporciona varios tipos de animación, siendo el más utilizado `Animated.timing()`. Admite la animación de un valor a lo largo del tiempo utilizando una de las diversas funciones de relajación predefinidas, o puede usar la suya propia. Las funciones de aceleración se utilizan normalmente en la animación para transmitir la aceleración y la desaceleración gradual de los objetos.
Por defecto, `timing` usará una curva easeInOut que transmite la aceleración gradual a velocidad máxima y concluye desacelerando gradualmente hasta detenerse. Puede especificar una función de relajación diferente pasando un `easing` parámetro. También se admite la costumbre `duration` o incluso una `delay` antes de que comience la animación.

Por ejemplo, si queremos crear una animación larga de 2 segundos de un objeto que retrocede un poco antes de moverse a su posición final:

```
Animated.timing(
        this.state.xPosition,
        {
                toValue: 100,
                easing: Easing.back,
                duration: 2000,
        }
).start();
```

Consulte la sección Configuración de animaciones de la `Animated` referencia de la API para obtener más información sobre todos los parámetros de configuración admitidos por las animaciones incorporadas.

Componer animaciones

Las animaciones se pueden combinar y reproducir en secuencia o en paralelo. Las animaciones secuenciales pueden reproducirse inmediatamente después de que la animación anterior haya finalizado, o pueden comenzar después de un retraso especificado. La `Animated` API proporciona varios métodos, como `sequence()` y `delay()`, cada uno de los cuales simplemente toma una matriz de animaciones para ejecutar y llama automáticamente `start()`/ `stop()`según sea necesario.

Por ejemplo, la siguiente animación se detiene por un costado, luego retrocede mientras gira en paralelo:

```
Animated.sequence([ // decay, then spring to start and twirl
        Animated.decay(position, { // coast to a stop
                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
```

Si una animación se detiene o se interrumpe, todas las demás animaciones del grupo también se detienen. `Animated.parallel` tiene una `stopTogether` opción que se puede configurar `false` para deshabilitar esto.

Puede encontrar una lista completa de métodos de composición en la sección Componer animaciones de la `Animated` referencia de la API.

Combinando valores animados

Puede combinar dos valores animados mediante suma, multiplicación, división o módulo para crear un nuevo valor animado.

Hay algunos casos en que un valor animado necesita invertir otro valor animado para el cálculo. Un ejemplo es invertir una escala (2x -> 0.5x):

```
const a = Animated.Value(1);
const b = Animated.divide(1, a);
```

```
Animated.spring(a, {
        toValue: 2,
}).start();
```

Interpolación

Cada propiedad puede ejecutarse primero a través de una interpolación. Una entrada de mapas de interpolación se extiende a los rangos de salida, generalmente utilizando una interpolación lineal, pero también admite funciones de relajación. Por defecto, extrapolará la curva más allá de los rangos dados, pero también puede hacer que fije el valor de salida.

Un mapeo simple para convertir un rango 0-1 a un rango de 0-100 sería:

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

Por ejemplo, puede considerar `Animated.Value` que va de 0 a 1, pero animar la posición de 150px a 0px y la opacidad de 0 a 1. Esto se puede hacer fácilmente modificando `style` el ejemplo anterior como se muestra a continuación:

```
style={{
        opacity: this.state.fadeAnim, // Binds directly
        transform: [{
                translateY: this.state.fadeAnim.interpolate({
                        inputRange: [0, 1],
                        outputRange: [150, 0] // 0 : 150, 0.5 : 75, 1 : 0
                }),
        }],
}}
```

`interpolate()` también admite segmentos de rango múltiple, lo que es útil para definir zonas muertas y otros trucos útiles. Por ejemplo, para obtener una relación de negación en -300 que va a 0 en -100, luego vuelva a 1 en 0, y luego vuelva a cero a 100 seguido de una zona muerta que permanece en 0 para todo más allá de eso, Podrías hacerlo:

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

Que se mapearía así:

```
Input | Output
------|-------
  -400| 450
  -300| 300
  -200| 150
  -100| 0
   -50| 0.5
     0| 1
    50| 0.5
   100| 0
   101| 0
   200| 0
```

`interpolate()` también admite el mapeo de cadenas, lo que le permite animar colores y valores con unidades. Por ejemplo, si quisieras animar una rotación, podrías hacer:

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

`interpolate()` también admite funciones de relajación arbitrarias, muchas de las cuales ya están implementadas en el `Easing`módulo.`interpolate()` también tiene un comportamiento configurable para extrapolar el `outputRange`. Puede configurar la extrapolación mediante el establecimiento de los `extrapolate`, `extrapolateLeft` o `extrapolateRight` las opciones. El valor predeterminado es `extend` pero puede usarse `clamp`para evitar que el valor de salida exceda `outputRange`.

Seguimiento de valores dinámicos

Los valores animados también pueden rastrear otros valores. Simplemente configure la `toValue` animación de otro valor animado en lugar de un número simple. Por ejemplo, una animación de "Cabezas de chat" como la utilizada por Messenger en Android podría implementarse con un `spring()` anclaje en otro valor animado, o con `timing()` y una `duration` de 0 para un seguimiento rígido. También se pueden componer con interpolaciones:

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

Los valores animados `leader` y `follower` se implementarían usando `Animated.ValueXY()`. `ValueXY` es una forma práctica de lidiar con las interacciones 2D, como el barrido o el arrastre. Es un envoltorio simple que básicamente contiene dos `Animated.Value` instancias y algunas funciones auxiliares que las llaman, haciendo `ValueXY` un reemplazo inmediato `Value` en muchos casos. Nos permite rastrear los valores xey en el ejemplo anterior.

Gestos de seguimiento

Los gestos, como la panorámica o desplazamiento, y otros eventos pueden asignarse directamente a los valores animados `Animated.event`. Esto se hace con una sintaxis de mapa estructurado para que los valores puedan extraerse de objetos de eventos complejos. El primer nivel es una matriz para permitir la asignación en múltiples argumentos, y esa matriz contiene objetos anidados.

Por ejemplo, cuando se trabaja con gestos de desplazamiento horizontal, debería hacer lo siguiente con el fin de asignar `event.nativeEvent.contentOffset.x` a `scrollX` (una `Animated.Value`):

```
onScroll={Animated.event(
        // scrollX = e.nativeEvent.contentOffset.x
        [{ nativeEvent: {
                        contentOffset: {
                                x: scrollX
                        }
                }
        }]
)}
```

Al usar `PanResponder`, puede usar el siguiente código para extraer las posiciones xey de `gestureState.dx` y `gestureState.dy`. Usamos un `null` en la primera posición de la matriz, ya que solo estamos interesados en el segundo argumento pasado al `PanResponder` controlador, que es el `gestureState`.

```
onPanResponderMove={Animated.event(
        [null, // ignore the native event
        // extract dx and dy from gestureState
        // like 'pan.x = gestureState.dx, pan.y = gestureState.dy'
        {dx: pan.x, dy: pan.y}
])}
```

Respondiendo al valor de animación actual

Puede observar que no hay una forma obvia de leer el valor actual mientras se anima. Esto se debe a que el valor solo se puede conocer en el tiempo de ejecución nativo debido a las optimizaciones. Si necesita ejecutar JavaScript en respuesta al valor actual, existen dos enfoques:

- `spring.stopAnimation(callback)` detendrá la animación e invocará `callback` con el valor final. Esto es útil al hacer transiciones de gestos.
- `spring.addListener(callback)` invocará `callback` asincrónicamente mientras la animación se está ejecutando, proporcionando un valor reciente. Esto es útil para activar cambios de estado, por ejemplo, ajustar un bobble a una nueva opción a medida que el usuario lo acerca, porque estos cambios de estado más grandes son menos sensibles a unos pocos fotogramas de retraso en comparación con gestos continuos como panning que deben ejecutarse a 60 fps

`Animated` está diseñado para ser completamente serializable, de modo que las animaciones se puedan ejecutar de una manera de alto rendimiento, independientemente del ciclo de eventos de JavaScript normal. Esto sí influye en la API, así que tenlo en cuenta cuando parezca un poco más complicado hacer algo en comparación con un sistema completamente sincrónico. Eche un vistazo `Animated.Value.addListener` como una forma de evitar algunas de estas limitaciones, pero úselo con moderación ya que podría tener implicaciones de rendimiento en el futuro.

Usando el controlador nativo

La `Animated` API está diseñada para ser serializable. Al usar el controlador nativo, enviamos todo lo relacionado con la animación a native antes de comenzar la animación, lo que permite que el código nativo realice la animación en el subproceso de la interfaz de usuario sin tener que pasar por el puente en cada fotograma. Una vez que la animación ha comenzado, el hilo JS se puede bloquear sin afectar la animación.

Usar el controlador nativo para animaciones normales es bastante simple. Solo agréguele `useNativeDriver: true` a la configuración de animación cuando lo inicie.

```
Animated.timing(this.state.animatedValue, {
        toValue: 1,
        duration: 500,
        useNativeDriver: true, // <-- Add this
}).start();
```

Los valores animados solo son compatibles con un controlador, por lo que si usa el controlador nativo al iniciar una animación en un valor, asegúrese de que cada animación en ese valor también use el controlador nativo.

El controlador nativo también trabaja con `Animated.event`. Esto es especialmente útil para las animaciones que siguen la posición de desplazamiento, ya que sin el controlador nativo, la animación siempre se ejecutará un fotograma detrás del gesto debido a la naturaleza asincrónica de React Native.

```
<Animated.ScrollView // <-- Use the Animated ScrollView wrapper
        scrollEventThrottle={1} // <-- Use 1 here to make sure no events are ever missed
        onScroll={Animated.event(
                [{ nativeEvent: { contentOffset: { y: this.state.animatedValue } } }],
                { useNativeDriver: true } // <-- Add this
        )}
>
        {content}
</Animated.ScrollView>
```

Puede ver el controlador nativo en acción ejecutando la aplicación RNTester y cargando el Ejemplo animado nativo. También puedes echarle un vistazo al código fuente para aprender cómo se produjeron estos ejemplos.

Advertencias

No todo lo que puede hacer con `Animated` es actualmente compatible con el controlador nativo. La principal limitación es que solo puede animar propiedades que no son de diseño: cosas como `transform` y `opacity` funcionarán, pero las propiedades de posición y de posición no lo harán. Al usarlo `Animated.event`, solo funcionará con eventos directos y no con eventos de burbujeo. Esto significa que no funciona, `PanResponder` pero funciona con cosas como `ScrollView#onScroll`.

Tenga en cuenta

Al usar estilos de transformación como `rotateY`, `rotateX` y otros, asegúrese de que el estilo de transformación `perspective` esté en su lugar. En este momento, es posible que algunas animaciones no se reproduzcan en Android sin este. Ejemplo a continuación.

```
<Animated.View
        style={{
                transform:
                        [
                                { scale: this.state.scale },
                                { rotateY: this.state.rotateY },
                                { perspective: 1000 } // without this line this Animation will not render on Android
while working fine on iOS
                        ]
                }}
        />
```

Ejemplos adicionales

La aplicación RNTester tiene varios ejemplos de `Animated` uso:

- AnimatedGratuitousApp

- NativeAnimationsEjemplo

`LayoutAnimation`API

`LayoutAnimation` le permite configurar globalmente `create` y `update` animaciones que se usarán para todas las vistas en el siguiente ciclo de renderizado / diseño. Esto es útil para realizar actualizaciones de diseño de flexbox sin molestarse en medir o calcular propiedades específicas para animarlas directamente, y es especialmente útil cuando los cambios de diseño pueden afectar a los antepasados, por ejemplo, una expansión "ver más" que también aumenta el tamaño del elemento primario y empuja hacia abajo la fila debajo de la cual, de lo contrario, se requiere una coordinación explícita entre los componentes para animarlos a todos en sincronización.
Tenga en cuenta que aunque `LayoutAnimation` es muy poderoso y puede ser bastante útil, proporciona mucho menos control que `Animated` otras bibliotecas de animación, por lo que puede necesitar usar otro enfoque si no puede `LayoutAnimation` hacer lo que desea.
Tenga en cuenta que para que esto funcione en **Android** , debe establecer los siguientes indicadores a través de `UIManager`:

```
UIManager.setLayoutAnimationEnabledExperimental &&
UIManager.setLayoutAnimationEnabledExperimental(true);
```

```
import React from 'react';
import {
  NativeModules,
  LayoutAnimation,
  Text,
  TouchableOpacity,
  StyleSheet,
  View,
} from 'react-native';

const { UIManager } = NativeModules;

UIManager.setLayoutAnimationEnabledExperimental &&
  UIManager.setLayoutAnimationEnabledExperimental(true);

export default class App extends React.Component {
  state = {
    w: 100,
    h: 100,
  };

  _onPress = () => {
    // Animate the update
    LayoutAnimation.spring();
    this.setState({w: this.state.w + 15, h: this.state.h + 15})
  }

  render() {
    return (
```

```
      <View style={styles.container}>
        <View style={[styles.box, {width: this.state.w, height: this.state.h}]} />
        <TouchableOpacity onPress={this._onPress}>
          <View style={styles.button}>
            <Text style={styles.buttonText}>Press me!</Text>
          </View>
        </TouchableOpacity>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  box: {
    width: 200,
    height: 200,
    backgroundColor: 'red',
  },
  button: {
    backgroundColor: 'black',
    paddingHorizontal: 20,
    paddingVertical: 15,
    marginTop: 15,
  },
  buttonText: {
    color: '#fff',
    fontWeight: 'bold',
  },
});
```

Este ejemplo utiliza un valor preestablecido, puede personalizar las animaciones según lo necesite, consulte LayoutAnimation.js para obtener más información.

Notas adicionales

`requestAnimationFrame`

`RequestAnimationFrame` es un relleno del navegador con el que puede estar familiarizado. Acepta una función como su único argumento y llama a esa función antes del próximo repintado. Es un elemento esencial para las animaciones que subyace a todas las API de animación basadas en JavaScript. En general, no debería necesitar llamarlo usted mismo: las API de animación administrarán las actualizaciones de cuadros por usted.

`setNativeProps`

Como se menciona en la sección Manipulación directa , `setNativeProps` nos permite modificar las propiedades de los componentes con respaldo nativo (componentes que en realidad están respaldados por vistas nativas, a diferencia de los componentes compuestos) directamente, sin tener que `setState`volver a renderizar la jerarquía de componentes.

Podríamos usar esto en el ejemplo de Rebote para actualizar la escala; esto podría ser útil si el componente que estamos actualizando está profundamente anidado y no ha sido optimizado `shouldComponentUpdate`.

Si encuentra sus animaciones con fotogramas caídos (con un rendimiento por debajo de 60 fotogramas por segundo), investigue cómo usarlos `setNativeProps` o `shouldComponentUpdate` optimizarlos. O puede ejecutar las animaciones en el subproceso de interfaz de usuario en lugar del subproceso de JavaScript con la opción useNativeDriver . También puede posponer cualquier trabajo intensivo desde el punto de vista computacional hasta que finalicen las animaciones, utilizando InteractionManager . Puede controlar la velocidad de fotogramas utilizando la herramienta "FPS Monitor" del menú del desarrollador en la aplicación.

Accesibilidad

Accesibilidad de aplicaciones nativas (iOS y Android)

Tanto iOS como Android proporcionan API para hacer que las personas con discapacidades tengan acceso a las aplicaciones. Además, ambas plataformas brindan tecnologías de asistencia agrupadas, como los lectores de pantalla VoiceOver (iOS) y TalkBack (Android) para las personas con discapacidad visual. Del mismo modo, en React Native hemos incluido API diseñadas para brindar a los desarrolladores asistencia para hacer que las aplicaciones sean más accesibles. Tome nota, iOS y Android difieren ligeramente en sus enfoques, y por lo tanto, las implementaciones de React Native pueden variar según la plataforma.

Además de esta documentación, puede encontrar esta publicación en el blogsobre la accesibilidad de React Native para ser útil.

Hacer aplicaciones accesibles

Propiedades de accesibilidad

accesible (iOS, Android)

Cuando `true`, indica que la vista es un elemento de accesibilidad. Cuando una vista es un elemento de accesibilidad, agrupa a sus hijos en un único componente seleccionable. Por defecto, todos los elementos tocables son accesibles.

En Android, la propiedad 'accessible = {true}' para una vista nativa a la reacción se traducirá a la versión nativa 'enfocable = {verdadero}'.

```
<View accessible={true}>
        <Text>text one</Text>
        <Text>text two</Text>
</View>
```

En el ejemplo anterior, no podemos obtener el enfoque de accesibilidad por separado en 'texto uno' y 'texto dos'. En cambio, obtenemos enfoque en una vista principal con propiedad 'accesible'.

accessibilityLabel (iOS, Android)

Cuando una vista se marca como accesible, es una buena práctica establecer un Label de accesibilidad en la vista, de modo que las personas que usan VoiceOver sepan qué elemento han seleccionado. VoiceOver leerá esta cadena cuando un usuario seleccione el elemento asociado.

Para usar, establezca la `accessibilityLabel` propiedad en una cadena personalizada en su Vista:

```
<TouchableOpacity accessible={true} accessibilityLabel={'Tap me!'} onPress={this._onPress}>
        <View style={styles.button}>
                <Text style={styles.buttonText}>Press me!</Text>
        </View>
</TouchableOpacity>
```

En el ejemplo anterior, el `accessibilityLabel` elemento TouchableOpacity tendría por defecto "¡Presione!". La etiqueta se construye concatenando todos los elementos del nodo Texto separados por espacios.

accesibilityTraits (iOS)

Los rasgos de accesibilidad le dicen a una persona que usa VoiceOver qué tipo de elemento ha seleccionado. ¿Este elemento es una etiqueta? ¿Un botón? Un encabezado? Estas preguntas son contestadas por `accessibilityTraits`.

Para usar, establezca la `accessibilityTraits` propiedad en una de (o una matriz de) cadenas de caracteres de accesibilidad:

- **ninguno** Se usa cuando el elemento no tiene rasgos.
- **botón** Se usa cuando el elemento debe tratarse como un botón.
- **enlace** Usado cuando el elemento debe tratarse como un enlace.
- **encabezado** Se usa cuando un elemento actúa como un encabezado para una sección de contenido (por ejemplo, el título de una barra de navegación).
- **búsqueda** Se usa cuando el elemento del campo de texto también debe tratarse como un campo de búsqueda.
- **imagen** Se usa cuando el elemento debe tratarse como una imagen. Se puede combinar con un botón o un enlace, por ejemplo.
- **seleccionado** Se usa cuando el elemento está seleccionado. Por ejemplo, una fila seleccionada en una tabla o un botón seleccionado dentro de un control segmentado.
- **juegos** Se usa cuando el elemento reproduce su propio sonido cuando está activado.
- **tecla** Se usa cuando el elemento actúa como una tecla del teclado.
- **texto** Se usa cuando el elemento debe tratarse como texto estático que no puede modificarse.
- **resumen** Se usa cuando un elemento se puede usar para proporcionar un resumen rápido de las condiciones actuales en la aplicación cuando se inicia por primera vez. Por ejemplo, cuando Weather se lanza por primera vez, el elemento con las condiciones climáticas actuales está marcado con este rasgo.
- **deshabilitado** Se usa cuando el control no está habilitado y no responde a la entrada del usuario.
- **frequentUpdates** Se usa cuando el elemento actualiza frecuentemente su etiqueta o valor, pero con demasiada frecuencia para enviar notificaciones. Permite a un cliente de accesibilidad realizar sondeos de cambios. Un cronómetro sería un ejemplo.
- **startsMedia** Se usa cuando la activación de un elemento inicia una sesión multimedia (por ejemplo, reproducir una película, grabar audio) que no debe interrumpirse por la salida de una tecnología de asistencia, como VoiceOver.
- **ajustable** Se usa cuando un elemento se puede "ajustar" (por ejemplo, un control deslizante).
- **allowDirectInteraction** Se usa cuando un elemento permite la interacción táctil directa para los usuarios de VoiceOver (por ejemplo, una vista que representa un teclado de piano).
- **pageTurn** informa a VoiceOver que debe desplazarse a la página siguiente cuando termine de leer el contenido del elemento.

accessibilityViewIsModal (iOS)

Un valor booleano que indica si VoiceOver debe ignorar los elementos dentro de las vistas que son hermanos del receptor.

Por ejemplo, en una ventana que contiene entre hermanos vistas `A`y `B`, fijando `accessibilityViewIsModal` a `true`la vista `B`hace que VoiceOver para ignorar los elementos de la vista `A`. Por otro lado, si la vista `B`contiene una vista del niño `C`y se establece `accessibilityViewIsModal` a `true` la vista `C`, VoiceOver no pasa por alto los elementos a la vista `A`.

onAccessibilityTap (iOS)

Utilice esta propiedad para asignar una función personalizada a la que se llamará cuando alguien active un elemento accesible al tocarla dos veces mientras está seleccionada.

onMagicTap (iOS)

Asigne esta propiedad a una función personalizada que se ejecutará cuando alguien realice el gesto de "toque mágico", que es un doble toque con dos dedos. Una función de toque mágico debería realizar la acción más

relevante que un usuario podría tomar sobre un componente. En la aplicación Teléfono en iPhone, un toque mágico responde una llamada telefónica o finaliza la actual. Si el elemento seleccionado no tiene una `onMagicTap` función, el sistema recorrerá la jerarquía de vista hasta que encuentre una vista que sí lo haga.

accessibilityComponentType (Android)

En algunos casos, también queremos alertar al usuario final del tipo de componente seleccionado (es decir, que es un "botón"). Si utilizáramos botones nativos, esto funcionaría automáticamente. Como usamos javascript, debemos proporcionar un poco más de contexto para TalkBack. Para hacerlo, debe especificar la propiedad 'accessibilityComponentType' para cualquier componente de UI. Por ejemplo, admitimos 'button', 'radiobutton_checked' y 'radiobutton_unchecked', y así sucesivamente.

```
<TouchableWithoutFeedback accessibilityComponentType=”button” onPress={this._onPress}>
        <View style={styles.button}>
                <Text style={styles.buttonText}>Press me!</Text>
        </View>
</TouchableWithoutFeedback>
```

En el ejemplo anterior, TalkBack está anunciando el Touchable sin retorno como un botón nativo.

accessibilityLiveRegion (Android)

Cuando los componentes cambian dinámicamente, queremos que TalkBack avise al usuario final. Esto es posible gracias a la propiedad 'accessibilityLiveRegion'. Se puede establecer en 'ninguno', 'cortés' y 'asertivo':

- **ninguno Los** servicios de accesibilidad no deben anunciar cambios a esta vista.
- **Los** servicios de Accesibilidad **cortés** deberían anunciar cambios a esta vista.
- **Los** servicios de accesibilidad **asertivos** deben interrumpir el discurso continuo para anunciar de inmediato los cambios a esta vista.

```
<TouchableWithoutFeedback onPress={this._addOne}>
        <View style={styles.embedded}>
                <Text>Click me</Text>
        </View>
</TouchableWithoutFeedback>
<Text accessibilityLiveRegion="polite">
        Clicked {this.state.count} times
</Text>
```

En el ejemplo anterior, el método _addOne cambia la variable state.count. Tan pronto como un usuario final haga clic en Touchable sin Feedback, TalkBack lee el texto en la vista de Texto debido a su propiedad 'accessibilityLiveRegion = "cortés"'.

importantForAccessibility (Android)

En el caso de dos componentes de interfaz de usuario superpuestos con el mismo elemento primario, el enfoque de accesibilidad predeterminado puede tener un comportamiento impredecible. La propiedad 'importantForAccessibility' resolverá esto al controlar si una vista dispara eventos de accesibilidad y si se informa a los servicios de accesibilidad. Se puede establecer en 'auto', 'yes', 'no' y 'no-hide-descenddants' (el último valor obligará a los servicios de accesibilidad a ignorar el componente y todos sus elementos secundarios).

```
<View style={styles.container}>
        <View style={{position: 'absolute', left: 10, top: 10, right: 10, height: 100, backgroundColor: 'green'}}
importantForAccessibility=”yes”>
```

```
                <Text> First layout </Text>
        </View>
        <View style={{position: 'absolute', left: 10, top: 10, right: 10, height: 100, backgroundColor: 'yellow'}}
importantForAccessibility=”no-hide-descendants”>
                <Text> Second layout </Text>
        </View>
</View>
```

En el ejemplo anterior, el diseño amarillo y sus descendientes son completamente invisibles para TalkBack y para todos los demás servicios de accesibilidad. Por lo tanto, podemos usar fácilmente vistas superpuestas con el mismo elemento primario sin confundir TalkBack.

Verificando si un lector de pantalla está habilitado

La `AccessibilityInfo` API le permite determinar si un lector de pantalla está actualmente activo o no. Consulte la documentación de AccessibilityInfo para más detalles.

Envío de eventos de accesibilidad (Android)

A veces es útil desencadenar un evento de accesibilidad en un componente de UI (es decir, cuando aparece una vista personalizada en una pantalla o se selecciona un botón de opción personalizado). El módulo UIManager nativo expone un método 'sendAccessibilityEvent' para este propósito. Toma dos argumentos: etiqueta de vista y un tipo de evento.

```
_onPress: function() {
        this.state.radioButton = this.state.radioButton === “radiobutton_checked” ? “radiobutton_unchecked” :
“radiobutton_checked”;
        if (this.state.radioButton === “radiobutton_checked” {
                RCTUIManager.sendAccessibilityEvent(
                        ReactNative.findNodeHandle(this),
                        RCTUIManager.AccessibilityEventTypes.typeViewClicked);
                }
        }
<CustomRadioButton
        accessibleComponentType={this.state.radioButton}
        onPress={this._onPress}/>
```

En el ejemplo anterior, hemos creado un botón de opción personalizado que ahora se comporta como uno nativo. Más específicamente, TalkBack ahora anuncia correctamente los cambios en la selección del botón de opción.

Probando el Soporte de VoiceOver (iOS)

Para habilitar VoiceOver, vaya a la aplicación Configuración en su dispositivo iOS. Toca General, luego Accesibilidad. Allí encontrará muchas herramientas que las personas usan para hacer que sus dispositivos sean más utilizables, como texto más audaz, mayor contraste y VoiceOver.

Para habilitar VoiceOver, toque VoiceOver en "Vision" y cambie el interruptor que aparece en la parte superior.

En la parte inferior de la configuración de Accesibilidad, hay un "Acceso directo de accesibilidad". Puede usar esto para alternar entre VoiceOver haciendo clic tres veces en el botón Inicio.

Mejorando la experiencia del usuario

La creación de aplicaciones para plataformas móviles se matiza, hay muchos pequeños detalles que los desarrolladores que provienen de un fondo web a menudo no consideran. Esta guía tiene la intención de explicar algunos de estos matices y demostrar cómo puede factorizarlos en su aplicación.

> Estamos mejorando y agregando más detalles a esta página. Si quieres ayudar, repítelo en react-native / 14979 .

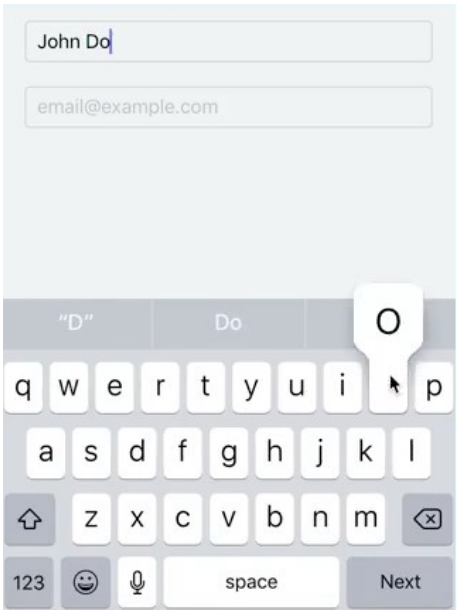
Índice de temas

Configurar entradas de texto

Ingresar texto en el teléfono táctil es un reto: pantalla pequeña, teclado de software. Pero según el tipo de datos que necesita, puede hacerlo más fácil configurando correctamente las entradas de texto:

- Enfoca el primer campo automáticamente
- Use texto de marcador de posición como un ejemplo del formato de datos esperado
- Habilitar o deshabilitar la autocapitalización y la autocorrección
- Elija el tipo de teclado (por ejemplo, correo electrónico, numérico)
- Asegúrese de que el botón de retorno enfoca el siguiente campo o envía el formulario

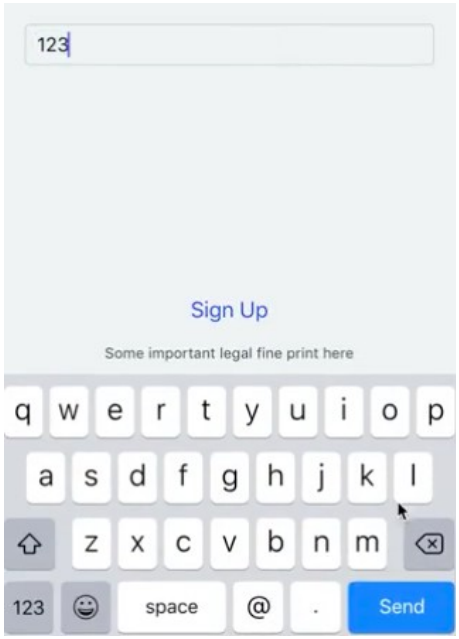
Consulte los `TextInput` documentos para obtener más opciones de configuración.



Pruébalo en tu teléfono

Administrar diseño cuando el teclado está visible

El teclado del software toma casi la mitad de la pantalla. Si tiene elementos interactivos que pueden cubrirse con el teclado, asegúrese de que todavía estén accesibles al usar `KeyboardAvoidingView`(ver documentos).



Pruébalo en tu teléfono

Hacer las áreas tappables más grandes

En los teléfonos móviles es difícil ser muy preciso al presionar los botones. Asegúrese de que todos los elementos interactivos tengan 44x44 o más. Una forma de hacerlo es dejar suficiente espacio para el elemento `padding`, `minWidth` y los `minHeight` valores de estilo pueden ser útiles para eso. Alternativamente, puede usar `hitSlop`prop para aumentar el área interactiva sin afectar el diseño. Aquí hay una demostración:

Pruébalo en tu teléfono

Use Android Ripple

La API de Android 21+ usa la onda de diseño del material para proporcionar retroalimentación al usuario cuando toca un área interactiva en la pantalla. React Native expone esto a través del `TouchableNativeFeedback` componente . El uso de este efecto táctil en lugar de opacidad o resaltado a menudo hará que su aplicación se sienta mucho más adecuada para la plataforma. Dicho esto, debe tener cuidado al usarlo, ya que no funciona en iOS o en Android API <21, por lo que tendrá que recurrir a uno de los otros componentes Touchable en iOS. Puede usar una biblioteca como react-native-platform-touchable para manejar las diferencias de la plataforma por usted.

Pruébalo en tu teléfono

Obtenga más información

Las Pautas de diseño de materiales e interfaz humana son excelentes recursos para aprender más sobre el diseño de plataformas móviles.

Timers

Los temporizadores son una parte importante de una aplicación y React Native implementa los temporizadores del navegador .

Timers

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

`requestAnimationFrame(fn)` no es lo mismo que `setTimeout(fn, 0)` - el primero disparará después de que todo el encuadre se haya enjuagado, mientras que el último disparará lo más rápido posible (más de 1000x por segundo en un iPhone 5S).

`setImmediate` se ejecuta al final del bloque de ejecución de JavaScript actual, justo antes de enviar la respuesta por lotes a su versión original. Tenga en cuenta que si llama `setImmediate` dentro de una `setImmediate` devolución de llamada, se ejecutará de inmediato, no cederá al nativo en el medio.

La `Promise` implementación usa `setImmediate` como primitiva asincrónica.

InteractionManager

Una razón por la cual las aplicaciones nativas bien construidas se sienten tan fluidas es evitando operaciones costosas durante interacciones y animaciones. En React Native, actualmente tenemos una limitación de que solo hay un único hilo de ejecución JS, pero puede usarlo `InteractionManager` para asegurarse de que el trabajo prolongado esté programado para comenzar después de que se hayan completado las interacciones / animaciones.

Las aplicaciones pueden programar tareas para que se ejecuten después de las interacciones con lo siguiente:

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

Compare esto con otras alternativas de programación:

- requestAnimationFrame (): para el código que anima una vista en el tiempo.
- setImmediate / setTimeout / setInterval (): ejecuta el código más tarde, tenga en cuenta que esto puede retrasar las animaciones.
- runAfterInteractions (): ejecuta el código más tarde, sin retrasar las animaciones activas.

El sistema de manejo táctil considera que uno o más toques activos son una "interacción" y retrasará las `runAfterInteractions()` devoluciones de llamadas hasta que todos los toques hayan finalizado o se hayan cancelado.

InteractionManager también permite que las aplicaciones registren animaciones mediante la creación de un "identificador" de interacción en el inicio de la animación y al borrarlo una vez completado:

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

TimerMixin

Descubrimos que la causa principal de muertes en las aplicaciones creadas con React Native se debió a que los temporizadores se activaron después de desmontar un componente. Para resolver este problema recurrente, presentamos `TimerMixin`. Si se incluye `TimerMixin`, a continuación, se puede reemplazar las llamadas a `setTimeout(fn, 500)` la `this.setTimeout(fn, 500)` (simplemente escriba `this.`) y todo se limpian adecuadamente para que cuando los componentes se desmonte.

Esta biblioteca no se envía con React Native: para usarlo en su proyecto, deberá instalarlo `npm i react-timer-mixin -save` desde el directorio de su proyecto.

```
import TimerMixin from 'react-timer-mixin';

var Component = createReactClass({
        mixins: [TimerMixin],
        componentDidMount: function() {
                this.setTimeout(
                        () => { console.log('I do not leak!');},
                        500
                );
        }
});
```

Esto eliminará una gran cantidad de trabajo duro rastreando errores, como fallas ocasionadas por el tiempo de activación después de que un componente ha sido desmontado.

Tenga en cuenta que si utiliza clases ES6 para sus componentes React, no hay una API incorporada para mixins . Para usar `TimerMixin` con clases ES6, recomendamos reaccionar-mixin .

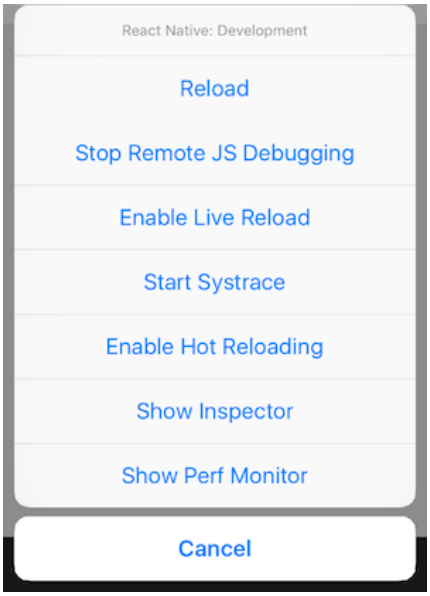
Depuración

Habilitación de atajos de teclado

React Native admite algunos atajos de teclado en el simulador de iOS. Se describen a continuación. Para habilitarlos, abre el menú Hardware, selecciona Teclado y asegúrate de que esté marcado "Conectar teclado de hardware".

Accediendo al menú del desarrollador en la aplicación

Puede acceder al menú del desarrollador sacudiendo su dispositivo o seleccionando "Agitar gesto" dentro del menú Hardware en el simulador de iOS. También puede usar el ⌘`D` atajo de teclado cuando su aplicación se ejecuta en el simulador de iOS o ⌘`M` cuando se ejecuta en un emulador de Android.



El menú del desarrollador está deshabilitado en versiones de lanzamiento (producción).

Recargando JavaScript

En lugar de recompilar su aplicación cada vez que realiza un cambio, puede volver a cargar el código JavaScript de su aplicación al instante. Para hacerlo, selecciona "Volver a cargar" en el Menú del desarrollador. También puede presionar ⌘`R`el simulador de iOS o tocar `R` dos veces en los emuladores de Android.

Recarga automática

Puede acelerar sus tiempos de desarrollo haciendo que su aplicación se vuelva a cargar automáticamente cada vez que cambie su código. La recarga automática se puede habilitar seleccionando "Habilitar recarga en vivo" desde el menú del desarrollador.

Incluso puede ir un paso más allá y mantener su aplicación en funcionamiento a medida que se inyectan nuevas versiones de sus archivos en el paquete de JavaScript automáticamente al habilitar la recarga en caliente desde el menú del desarrollador. Esto le permitirá persistir el estado de la aplicación a través de recargas.

> En algunos casos, la recarga en caliente no se puede implementar a la perfección. Si se encuentra con algún problema, use una recarga completa para restablecer su aplicación.

Necesitarás reconstruir tu aplicación para que los cambios surtan efecto en ciertas situaciones:

- Ha agregado recursos nuevos al paquete de su aplicación nativa, como una imagen en `Images.xcassets` iOS o la `res/drawable` carpeta en Android.
- Has modificado el código nativo (Objective-C / Swift en iOS o Java / C ++ en Android).

Errores y advertencias en la aplicación

Los errores y advertencias se muestran dentro de su aplicación en compilaciones de desarrollo.

Errores

Los errores en la aplicación se muestran en una alerta de pantalla completa con un fondo rojo dentro de la aplicación. Esta pantalla se conoce como RedBox. Puede usar `console.error()` para activar manualmente uno.

Advertencias

Las advertencias se mostrarán en la pantalla con un fondo amarillo. Estas alertas se conocen como YellowBoxes.

Haga clic en las alertas para mostrar más información o descartarlas.

Al igual que con RedBox, puede usar `console.warn()` para activar un YellowBox.

YellowBoxes se puede desactivar durante el desarrollo mediante `console.disableYellowBox=true;`.

Advertencias específicas pueden ser ignorados mediante programación estableciendo un conjunto de prefijos que debe ser ignorado: `console.ignoredYellowBox = ['Warning: ...'];`.

En CI / Xcode, YellowBoxes también se puede desactivar configurando la `IS_TESTING` variable de entorno.

> RedBoxes y YellowBoxes se desactivan automáticamente en versiones de lanzamiento (producción).

Herramientas de desarrollo de Chrome

Para depurar el código JavaScript en Chrome, seleccione "Depurar JS Remotamente" en el Menú del Desarrollador. Esto abrirá una nueva pestaña en http: // localhost: 8081 / debugger-ui .

Seleccione `Tools → Developer Tools` desde el Menú de Chrome para abrir las Herramientas de desarrollo .

También puede acceder a DevTools usando atajos de teclado (`⌘⌥I`en macOS, `Ctrl Shift I`en Windows).

También es posible que desee habilitar Pausa en excepciones capturadas para una mejor experiencia de depuración.

> Nota: la extensión de React Developer Tools Chrome no funciona con React Native, pero puedes usar su versión independiente. Lee esta sección para aprender cómo.

Depuración utilizando un depurador de JavaScript personalizado

Para usar un depurador de JavaScript personalizado en lugar de las Herramientas de desarrollador de Chrome, establezca la `REACT_DEBUGGER` variable de entorno en un comando que iniciará su depurador personalizado. A continuación, puede seleccionar "Depurar JS Remotamente" en el Menú del Desarrollador para iniciar la depuración.

El depurador recibirá una lista de todas las raíces del proyecto, separadas por un espacio. Por ejemplo, si configura `REACT_DEBUGGER="node /path/to/launchDebugger.js --port 2345 --type ReactNative"`, el comando `node /path/to/launchDebugger.js --port 2345 --type ReactNative /path/to/reactNative/app` se usará para iniciar su depurador.

> Los comandos de depurador personalizados ejecutados de esta manera deben ser procesos de corta duración, y no deben producir más de 200 kilobytes de salida.

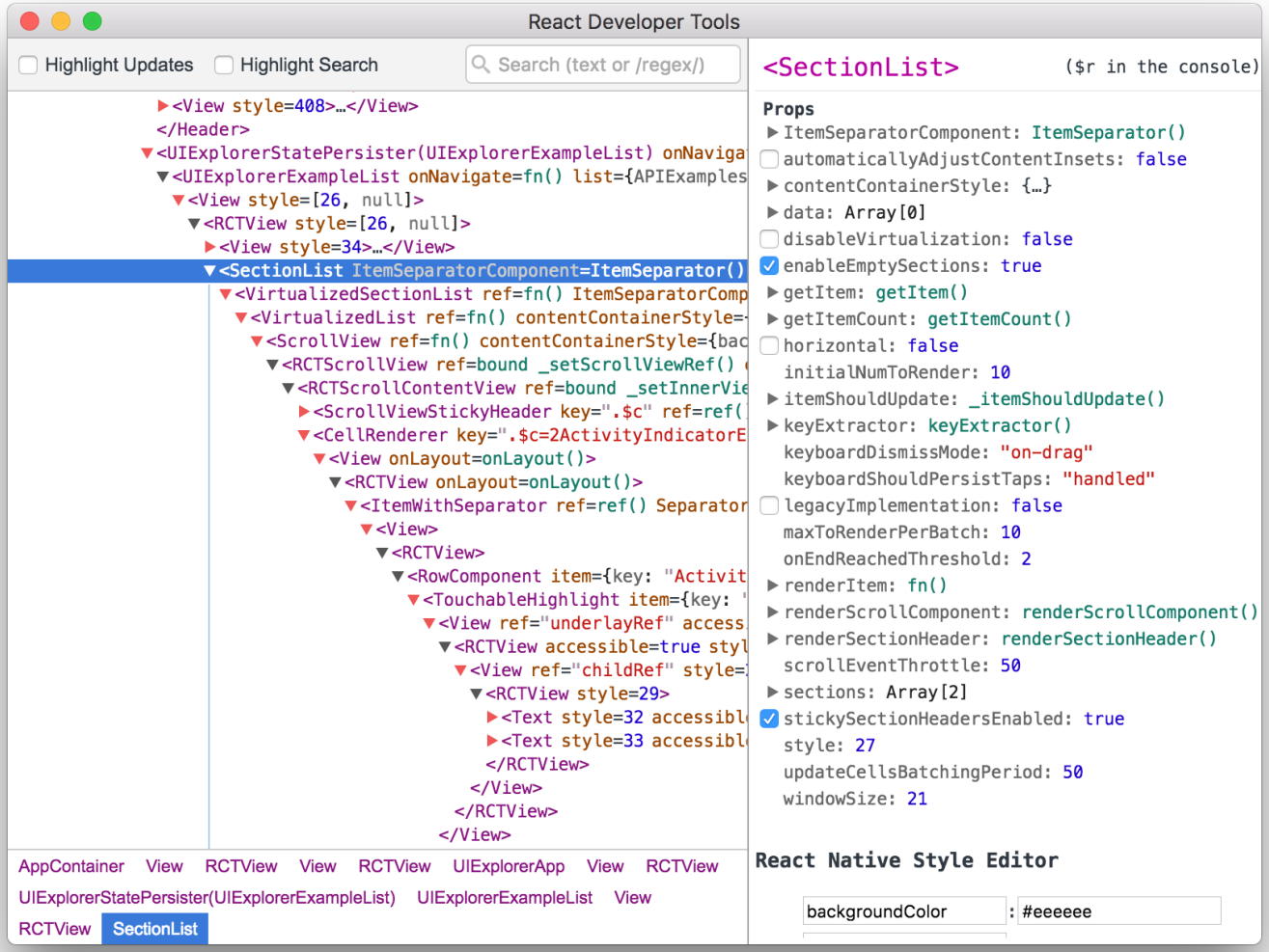
React herramientas de desarrollo

Puede usar la versión independiente de React Developer Tools para depurar la jerarquía de componentes Reaccionar. Para usarlo, instale el `react-devtools` paquete globalmente:

```
npm install -g react-devtools
```

Ahora ejecute `react-devtools` desde la terminal para iniciar la aplicación DevTools independiente:

```
react-devtools
```

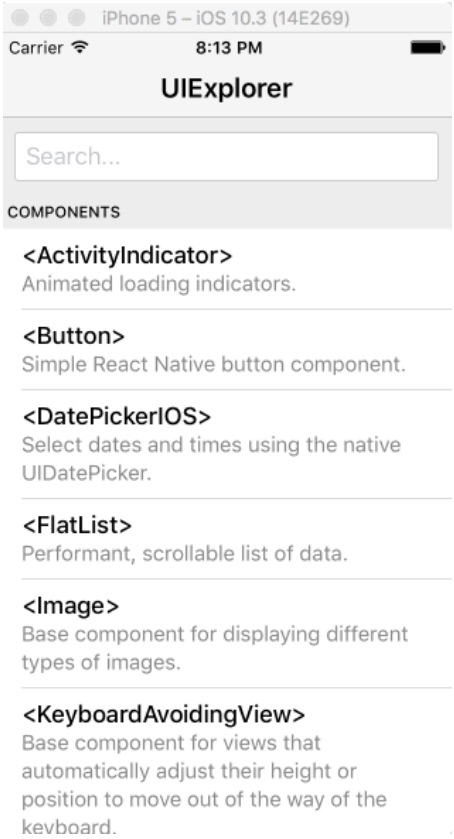


Debería conectarse a su simulador en unos pocos segundos.

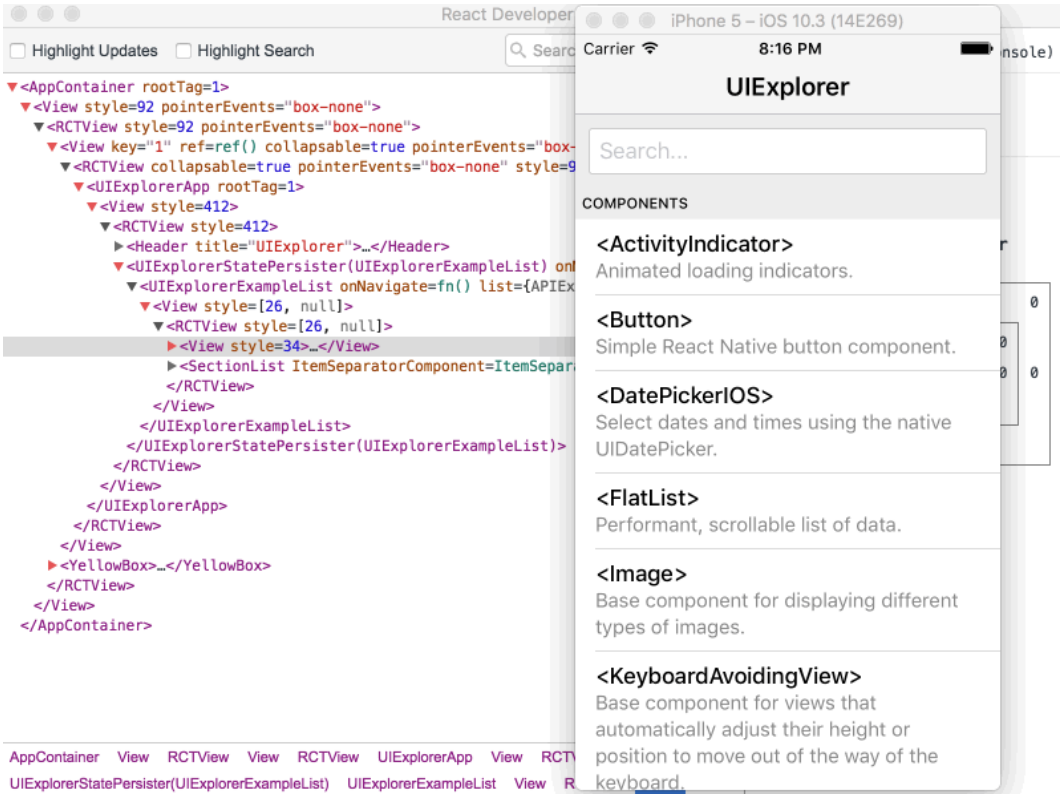
> Nota: si prefiere evitar instalaciones globales, puede agregar `react-devtools` como dependencia del proyecto. Agregue el `react-devtools` paquete a su proyecto usando `npm install --save-dev react-devtools`, luego agregue `"react-devtools": "react-devtools"` a la `scripts` sección en su `package.json`, y luego ejecútelo `npm run react-devtools` desde su carpeta de proyectos para abrir DevTools.

Integración con React Native Inspector

Abra el menú del desarrollador en la aplicación y elija "Mostrar inspector". Aparecerá una superposición que le permite tocar cualquier elemento de la interfaz de usuario y ver información al respecto:



Sin embargo, cuando se `react-devtools` está ejecutando, el Inspector ingresará a un modo colapsado especial y, en su lugar, usará DevTools como UI principal. En este modo, al hacer clic en algo en el simulador se mostrarán los componentes relevantes en DevTools:



Puede elegir "Ocultar inspector" en el mismo menú para salir de este modo.

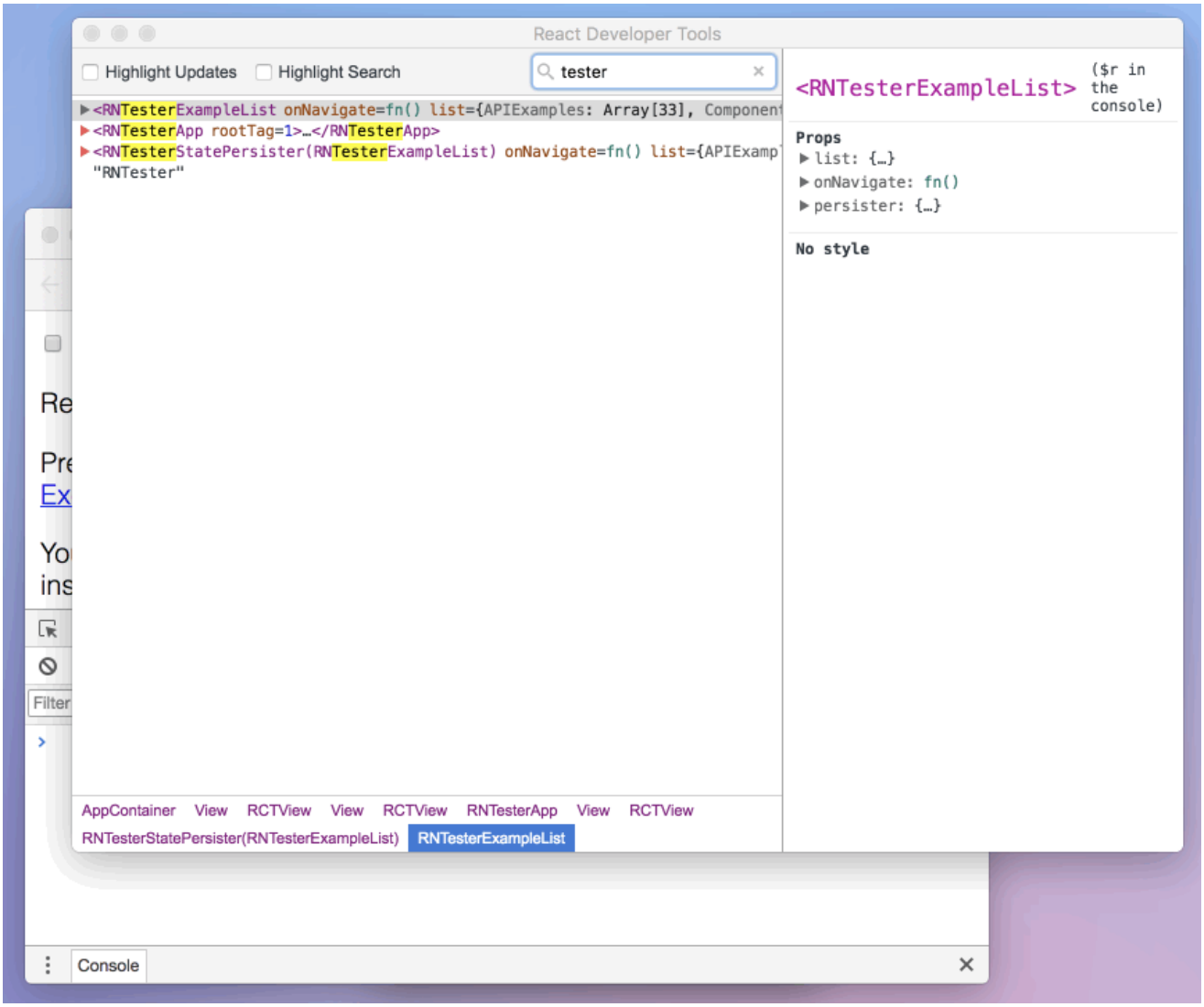
Inspección de instancias de componentes

Al depurar JavaScript en Chrome, puede inspeccionar los accesorios y el estado de los componentes de React en la consola del navegador.

Primero, sigue las instrucciones para depurar en Chrome para abrir la consola de Chrome.

Asegúrate de que aparezca el menú desplegable en la esquina superior izquierda de la consola de Chrome `debuggerWorker.js`. **Este paso es esencial.**

Luego seleccione un componente React en React DevTools. Hay un cuadro de búsqueda en la parte superior que te ayuda a encontrar uno por nombre. Tan pronto como lo seleccione, estará disponible como `$r` en la consola de Chrome, lo que le permitirá inspeccionar sus atributos, estado y propiedades de instancia.



Monitor de rendimiento

Puede habilitar una superposición de rendimiento para ayudarlo a depurar problemas de rendimiento seleccionando "Perf Monitor" en el Menú de Desarrollador.

Depuración en aplicaciones ejecutadas

Proyectos con código nativo solamente

El resto de este capítulo solo se aplica a proyectos realizados con `react-native init` o creados con la aplicación nativa Create React que se han expulsado. Para obtener más información sobre la expulsión, consulte la guíaen el repositorio Crear repositorio nativo de la aplicación.

Accediendo a los registros de la consola

Puede visualizar los registros de la consola para una aplicación iOS o Android utilizando los siguientes comandos en un terminal mientras la aplicación se está ejecutando:

```
$ react-native log-ios
$ react-native log-android
```

También puede acceder a ellos `Debug → Open System Log...` en el simulador de iOS o al ejecutar `adb logcat *:S ReactNative:V ReactNativeJS:V` en un terminal mientras se ejecuta una aplicación de Android en un dispositivo o emulador.

> Si está utilizando la aplicación Create React Native, los registros de la consola ya aparecen en la misma salida de terminal que el empaquetador.

Depuración en un dispositivo con Chrome Developer Tools

> Si está utilizando la aplicación Create React Native, esto ya está configurado para usted.

En los dispositivos iOS, abra el archivo `RCTWebSocketExecutor.m` y cambie "localhost" a la dirección IP de su computadora, luego seleccione "Depurar JS Remotely" desde el Menú del Desarrollador.

En dispositivos Android 5.0+ conectados a través de USB, puede usar la `adb`herramienta de línea de comandos para configurar el reenvío de puertos desde el dispositivo a su computadora:

```
adb reverse tcp:8081 tcp:8081
```

Alternativamente, seleccione "Configuraciones Dev" en el Menú del Desarrollador, luego actualice la configuración "Servidor del servidor de depuración para el dispositivo" para que coincida con la dirección IP de su computadora.

> Si se encuentra con algún problema, es posible que una de sus extensiones de Chrome interactúe de forma inesperada con el depurador. Intente deshabilitar todas sus extensiones y vuelva a habilitarlas una a una hasta que encuentre la extensión problemática.

Depuración con Stetho en Android

Siga esta guía para habilitar Stetho para el modo de depuración:

1. En `android/app/build.gradle`, agregue estas líneas en la `dependencies` sección:

```
debugCompile 'com.facebook.stetho:stetho:1.5.0'
debugCompile 'com.facebook.stetho:stetho-okhttp3:1.5.0'
```

> Lo anterior configurará Stetho v1.5.0. Puede consultar en http://facebook.github.io/stetho/ si hay una versión más nueva disponible.

1. Cree las siguientes clases de Java para envolver la llamada Stetho, una para el lanzamiento y otra para la depuración:

```
// android/app/src/release/java/com/{yourAppName}/StethoWrapper.java

public class StethoWrapper {

        public static void initialize(Context context) {
                // NO_OP
        }

        public static void addInterceptor() {
                // NO_OP
        }
```

```
}


// android/app/src/debug/java/com/{yourAppName}/StethoWrapper.java

public class StethoWrapper {

        public static void initialize(Context context) {
                Stetho.initializeWithDefaults(context);
        }

        public static void addInterceptor() {
                OkHttpClient client = OkHttpClientProvider.getOkHttpClient()
                .newBuilder()
                .addNetworkInterceptor(new StethoInterceptor())
                .build();

                OkHttpClientProvider.replaceOkHttpClient(client);
        }
}
```

2. Abra `android/app/src/main/java/com/{yourAppName}/MainApplication.java` y reemplace la `onCreate` función original :

```
public void onCreate() {
        super.onCreate();
        if (BuildConfig.DEBUG) {
                StethoWrapper.initialize(this);
                StethoWrapper.addInterceptor();
        }

        SoLoader.init(this, /* native exopackage */ false);
}
```

1. Abra el proyecto en Android Studio y resuelva cualquier problema de dependencia. El IDE debe guiarte por estos pasos después de pasar el puntero sobre las líneas rojas.
2. Ejecutar `react-native run-android`.
3. En una nueva pestaña de Chrome, abra : `chrome://inspect`, luego haga clic en el elemento 'Inspeccionar dispositivo' junto a 'Desarrollado por Stetho'.
4.

Depuración del código nativo

Al trabajar con código nativo, como al escribir módulos nativos, puede iniciar la aplicación desde Android Studio o Xcode y aprovechar las características de depuración nativa (configuración de puntos de interrupción, etc.) como lo haría en el caso de construir una aplicación nativa estándar .

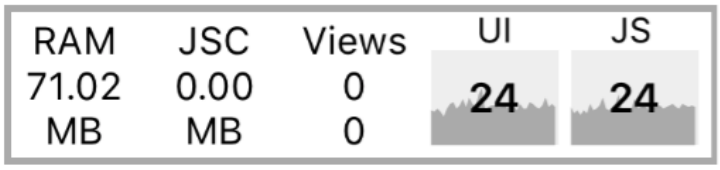
Rendimiento

Una razón convincente para usar React Native en lugar de herramientas basadas en WebView es lograr 60 cuadros por segundo y una apariencia nativa para sus aplicaciones. Siempre que sea posible, nos gustaría que React Native haga lo correcto y lo ayude a centrarse en su aplicación en lugar de optimizar el rendimiento, pero hay áreas en las que aún no llegamos, y otras en React Native (similar a la escritura nativa). código directamente) no es posible determinar la mejor manera de optimizar para usted y por lo tanto será necesaria la intervención manual. Hacemos nuestro mejor esfuerzo para ofrecer un rendimiento UI suave como la mantequilla por defecto, pero a veces eso simplemente no es posible.

Esta guía tiene la intención de enseñarle algunos conceptos básicos para ayudarlo a solucionar problemas de rendimiento , así como para analizar fuentes comunes de problemas y sus soluciones sugeridas .

Lo que necesita saber sobre marcos

La generación de tus abuelos llamó a las películas "imágenes en movimiento" por una razón: el movimiento realista en video es una ilusión creada al cambiar rápidamente las imágenes estáticas a una velocidad constante. Nos referimos a cada una de estas imágenes como marcos. La cantidad de fotogramas que se muestran cada segundo tiene un impacto directo en la suavidad y, en última instancia, en la vida de un video (o interfaz de usuario). Los dispositivos con iOS muestran 60 fotogramas por segundo, lo que le proporciona a usted y al sistema de IU alrededor de 16.67ms para hacer todo el trabajo necesario para generar la imagen estática (fotograma) que el usuario verá en la pantalla durante ese intervalo. Si no puede hacer el trabajo necesario para generar ese marco dentro de los 16.67ms asignados, entonces "soltará un marco" y la interfaz de usuario parecerá que no responde. Ahora, para confundir el asunto un poco, abra el menú del desarrollador en su aplicación y alternar `Show Perf Monitor`. Notará que hay dos tasas de cuadros diferentes.



Tasa de cuadros JS (hilo de JavaScript)

Para la mayoría de las aplicaciones de React Native, su lógica comercial se ejecutará en el hilo de JavaScript. Aquí es donde vive su aplicación Reaccionar, se realizan llamadas a la API, se procesan los eventos táctiles, etc. Las actualizaciones de las vistas con respaldo nativo se agrupan y se envían al lado nativo al final de cada iteración del ciclo de evento, antes de la fecha límite del marco (si todo va bien). Si el subproceso JavaScript no responde a un marco, se considerará un marco descartado. Por ejemplo, si llamaras `this.setState` en el componente raíz de una aplicación compleja y que resultó en la reutilización de subárboles de componentes computacionalmente costosos, es concebible que esto pueda tomar 200 ms y dar como resultado la pérdida de 12 fotogramas. Cualquier animación controlada por JavaScript parece congelarse durante ese tiempo. Si algo lleva más de 100 ms, el usuario lo sentirá.

Esto sucede a menudo durante las `Navigator` transiciones: cuando se empuja una nueva ruta, el hilo JavaScript necesita representar todos los componentes necesarios para la escena a fin de enviar los comandos adecuados al

lado nativo para crear las vistas de respaldo. Es común que el trabajo que se realiza aquí tome algunos fotogramas y cause jank porque la transición está controlada por el hilo de JavaScript. A veces, los componentes harán un trabajo adicional `componentDidMount`, lo que podría ocasionar un segundo tartamudeo en la transición.
Otro ejemplo es responder a los toques: si está trabajando en varios fotogramas en el hilo de JavaScript, es posible que observe un retraso en la respuesta `TouchableOpacity`, por ejemplo. Esto se debe a que el hilo JavaScript está ocupado y no puede procesar los eventos táctiles sin procesar enviados desde el hilo principal. Como resultado, `TouchableOpacity`no puede reaccionar a los eventos táctiles y ordena la vista nativa para ajustar su opacidad.

Velocidad de cuadro UI (hilo principal)

Muchas personas han notado que el rendimiento `NavigatorIOS` es mejor de lo que está `Navigator`. La razón de esto es que las animaciones para las transiciones se realizan por completo en el hilo principal, por lo que no se interrumpen mediante la reducción de cuadros en el hilo de JavaScript.
Del mismo modo, puedes desplazarte hacia arriba y hacia abajo con facilidad `ScrollView` cuando el hilo de JavaScript está bloqueado porque las `ScrollView` vidas se encuentran en el hilo principal. Los eventos de desplazamiento se envían al subproceso JS, pero su recepción no es necesaria para que se produzca el desplazamiento.

Fuentes comunes de problemas de rendimiento

Corriendo en modo de desarrollo (`dev=true`)

El rendimiento del subproceso de JavaScript sufre mucho cuando se ejecuta en modo dev. Esto es inevitable: se necesita mucho más trabajo en tiempo de ejecución para proporcionarle buenas advertencias y mensajes de error, como la validación de propTypes y varias otras afirmaciones. Siempre asegúrese de probar el rendimiento en compilaciones de lanzamiento .

Usando `console.log` declaraciones

Cuando se ejecuta una aplicación integrada, estas declaraciones pueden causar un gran cuello de botella en el hilo de JavaScript. Esto incluye llamadas de bibliotecas de depuración como redux-logger , así que asegúrese de eliminarlas antes de agruparlas. También puede usar este complemento Babel que elimina todas las `console.*` llamadas. Primero debe instalarlo `npm i babel-plugin-transform-remove-console -save` y luego edite el `.babelrc` archivo en su directorio de proyecto de la siguiente manera:

```
{
        "env": {
                "production": {
                        "plugins": ["transform-remove-console"]
                }
        }
}
```

Esto eliminará automáticamente todas las `console.*` llamadas en las versiones de lanzamiento (producción) de su proyecto.

`ListView` la representación inicial es demasiado lenta o el rendimiento de desplazamiento es malo para listas grandes

Use el nuevo `FlatList`o `SectionList`componente en su lugar. Además de simplificar la API, los nuevos componentes de la lista también tienen importantes mejoras de rendimiento, la principal es el uso de la memoria casi constante para cualquier cantidad de filas.

Si su `FlatList` renderizado es lento, asegúrese de haberlo implementado `getItemLayout` para optimizar la velocidad de representación omitiendo la medición de los elementos renderizados.

JS FPS se sumerge al volver a renderizar una vista que apenas cambia

Si está utilizando un ListView, debe proporcionar una `rowHasChanged` función que pueda reducir mucho trabajo al determinar rápidamente si una fila necesita volver a procesarse o no. Si está utilizando estructuras de datos inmutables, esto sería tan simple como una verificación de igualdad de referencia.

Del mismo modo, puede implementar `shouldComponentUpdate` e indicar las condiciones exactas bajo las cuales desea que el componente vuelva a renderizarse. Si escribe componentes puros (donde el valor de retorno de la función de renderización depende completamente de los apoyos y el estado), puede aprovechar PureRenderMixin para hacer esto por usted. Una vez más, las estructuras de datos inmutables son útiles para mantener esto rápido: si tiene que hacer una comparación profunda de una gran lista de objetos, es posible que volver a procesar su componente completo sea más rápido y sin duda requerirá menos código. .

Eliminando FPS de subproceso JS porque hace mucho trabajo en el subproceso de JavaScript al mismo tiempo

"Transiciones del navegador lento" es la manifestación más común de esto, pero hay otras veces que esto puede suceder. El uso de InteractionManager puede ser un buen enfoque, pero si el costo de la experiencia del usuario es demasiado alto para retrasar el trabajo durante una animación, es posible que desee considerar LayoutAnimation.

La API animada calcula actualmente cada fotograma clave según demanda en el hilo de JavaScript a menos que lo configure`useNativeDriver: true` , mientras que LayoutAnimation aprovecha Core Animation y no se ve afectado por el hilo JS y las caídas de marco de hilo principal.

Un caso donde lo he usado es para animar en un modal (deslizándose desde arriba y desvaneciéndose en una superposición translúcida) mientras se inicializan y quizás reciben respuestas para varias solicitudes de red, visualizando los contenidos del modal, y actualizando la vista donde el modal fue abierto desde. Consulte la guía de animaciones para obtener más información sobre cómo usar LayoutAnimation.

Advertencias:

- LayoutAnimation solo funciona para las animaciones "fire-and-forget" (animaciones "estáticas"): si debe ser interrumpible, deberá usarla `Animated`.

Mover una vista en la pantalla (desplazarse, traducir, rotar) deja caer el hilo de la interfaz de usuario FPS

Esto es especialmente cierto cuando tiene texto con un fondo transparente colocado en la parte superior de una imagen, o en cualquier otra situación en la que se requiera una composición alfa para volver a dibujar la vista en cada cuadro. Encontrará que habilitar `shouldRasterizeIOS` o `renderToHardwareTextureAndroid` puede ayudar con esto significativamente.

Tenga cuidado de no abusar de esto o su uso de memoria podría ir por las nubes. Perfile su rendimiento y uso de la memoria cuando use estos accesorios. Si no planea mover una vista más, desactive esta propiedad.

Animar el tamaño de una imagen deja caer el hilo UI FPS

En iOS, cada vez que ajusta el ancho o alto de un componente de imagen, se recorta y escala a partir de la imagen original. Esto puede ser muy costoso, especialmente para imágenes grandes. En su lugar, use la `transform: [{scale}]` propiedad de estilo para animar el tamaño. Un ejemplo de cuándo puede hacer esto es cuando toca una imagen y la acerca a pantalla completa.

Mi vista TouchableX no es muy sensible

A veces, si hacemos una acción en el mismo marco en el que estamos ajustando la opacidad o el resaltado de un componente que responde a un toque, no veremos ese efecto hasta después de que la `onPress` función haya regresado. Si `onPress` hace una `setState` que da lugar a una gran cantidad de trabajo y unos pocos fotogramas, esto puede ocurrir. Una solución a esto es ajustar cualquier acción dentro de su `onPress` controlador en `requestAnimationFrame`:

```
handleOnPress() {
        // Always use TimerMixin with requestAnimationFrame, setTimeout and
        // setInterval
        this.requestAnimationFrame(() => {
                this.doExpensiveAction();
        });
}
```

Transiciones del navegador lento

Como se mencionó anteriormente, las `Navigator` animaciones están controladas por el hilo de JavaScript. Imagine la transición de escena "empujar desde la derecha": cada fotograma, la nueva escena se mueve de derecha a izquierda, comienza fuera de pantalla (digamos en un x-offset de 320) y finalmente se establece cuando la escena se encuentra en un x-offset de 0. Cada cuadro durante esta transición, el hilo de JavaScript necesita enviar un nuevo x-offset al hilo principal. Si el hilo de JavaScript está bloqueado, no puede hacer esto y no se produce ninguna actualización en ese marco y la animación tartamudea.

Una solución para esto es permitir que las animaciones basadas en JavaScript se descarguen al hilo principal. Si tuviéramos que hacer lo mismo que en el ejemplo anterior con este enfoque, podríamos calcular una lista de todas las x-offsets para la nueva escena cuando estamos comenzando la transición y enviarlas al hilo principal para que se ejecuten de manera optimizada . Ahora que el hilo de JavaScript se libera de esta responsabilidad, no es gran cosa si deja caer unos pocos cuadros mientras renderiza la escena; probablemente ni siquiera lo notarás porque estarás demasiado distraído por la bonita transición.

Resolver esto es uno de los principales objetivos detrás de la nueva biblioteca React Navigation . Las vistas en React Navigation usan componentes nativos y la `Animated`biblioteca para entregar 60 animaciones FPS que se ejecutan en el hilo nativo.

Perfil

Utilice el generador de perfiles integrado para obtener información detallada sobre el trabajo realizado en el hilo de JavaScript y en el hilo principal, uno al lado del otro. Acceda a él seleccionando Perf Monitor en el menú Depurar. Para iOS, Instruments es una herramienta invaluable, y en Android debes aprender a usarla `systrace`.

También puede usar `react-addons-perf`para obtener información sobre dónde está gastando el tiempo React al representar sus componentes.

Otra forma de crear un perfil de JavaScript es usar el perfil de Chrome durante la depuración. Esto no le dará resultados precisos ya que el código se está ejecutando en Chrome, pero le dará una idea general de dónde podrían haber cuellos de botella.

Pero primero, ¡ **asegúrate de que el Modo de Desarrollo esté APAGADO!** Debería ver `__DEV__ === false, development-level warning are OFF, performance optimizations are ON` en los registros de su aplicación.

Perfilando el rendimiento de la interfaz de usuario de Android con `systrace`

Android admite 10k + teléfonos diferentes y se generaliza para admitir el procesamiento de software: la arquitectura de marco y la necesidad de generalizar a través de muchos objetivos de hardware desafortunadamente significa que obtiene menos de forma gratuita en relación con iOS. Pero a veces, hay cosas que puedes mejorar, ¡y muchas veces no es culpa del código nativo!

El primer paso para depurar este jank es responder a la pregunta fundamental de dónde se gasta su tiempo durante cada cuadro de 16ms. Para eso, utilizaremos una herramienta de generación de perfiles estándar de Android `systrace`.

`Systrace` es una herramienta estándar de generación de perfiles basada en marcadores de Android (y se instala cuando instala el paquete de herramientas de la plataforma Android). Los bloques de código perfilados están rodeados por marcadores de inicio / final que luego se visualizan en un formato de gráfico colorido. Tanto el marco de Android SDK como el de React Native proporcionan marcadores estándar que puedes visualizar.

1. Recopilando un rastro

Primero, conecte un dispositivo que muestre el tartamudeo que desea investigar a su computadora a través de USB y llévelo al punto justo antes de la navegación / animación que desea crear. Ejecuta de la `systrace` siguiente manera:

```
$ <path_to_android_sdk>/platform-tools/systrace/systrace.py --time=10 -o trace.html sched gfx view -a
<your_package_name>
```

Un desglose rápido de este comando:

- `time` es la cantidad de tiempo que la traza se recogerá en segundos
- `sched`, `gfx` y `view` son las etiquetas SDK de Android (colecciones de marcadores) que nos importan: `sched`le brinda información sobre lo que se está ejecutando en cada núcleo de su teléfono, `gfx`le brinda información gráfica, como límites de marcos, y `view`le brinda información sobre medidas, diseño y dibujo pasa
- `-a <your_package_name>` permite marcadores específicos de la aplicación, específicamente los incorporados en el marco Reaccionar nativo. `your_package_name` se puede encontrar en la `AndroidManifest.xml` aplicación y se ve como `com.example.app`

Una vez que la traza comienza a recopilarse, realice la animación o interacción que le interese. Al final del seguimiento, systrace le dará un enlace al seguimiento que puede abrir en su navegador.

2. Leyendo el rastro

Después de abrir el seguimiento en su navegador (preferiblemente Chrome), debería ver algo como esto:

SUGERENCIA : Use las teclas WASD para atacar y hacer zoom

Si su archivo .html de rastreo no se abre correctamente, consulte la consola de su navegador para ver lo siguiente:

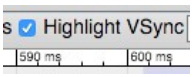
```
Uncaught TypeError: Object.observe is not a function
```

Como ya no `Object.observe` está disponible en los navegadores recientes, es posible que deba abrir el archivo desde la herramienta de seguimiento de Google Chrome. Puedes hacerlo así:

- Pestaña de apertura en chrome chrome: // tracing
- Seleccionando carga
- Seleccionar el archivo html generado desde el comando anterior.

Habilitar resaltado VSync

Marque esta casilla en la parte superior derecha de la pantalla para resaltar los límites del cuadro de 16ms:

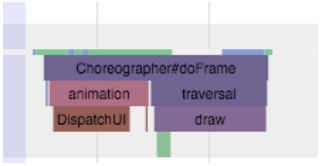


Debería ver rayas de cebra como en la captura de pantalla anterior. Si no lo hace, intente crear un perfil en un dispositivo diferente: se sabe que Samsung tiene problemas para mostrar vsyncs, mientras que la serie Nexus generalmente es bastante confiable.

3. Encuentra tu proceso

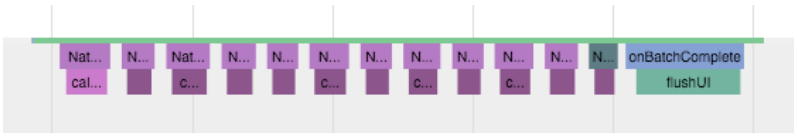
Desplácese hasta que vea (parte de) el nombre de su paquete. En este caso, estaba `com.facebook.adsmanager` haciendo un perfil , que aparece como `book.adsmanager` debido a los límites del nombre del hilo en el kernel. En el lado izquierdo, verá un conjunto de hilos que corresponden a las filas de la línea de tiempo a la derecha. Hay algunos hilos que nos importan para nuestros propósitos: el hilo de UI (que tiene el nombre de su paquete o el nombre UI Thread) `mqt_js`, y `mqt_native_modules`. Si está ejecutando Android 5+, también nos preocupamos por el hilo Render.

- **UI Thread.** Aquí es donde sucede la medida / diseño / sorteo estándar de Android. El nombre del hilo a la derecha será su nombre de paquete (en mi caso book.adsmanager) o UI Thread. Los eventos que se ven en este hilo debe ser algo como esto y tienen que ver con `Choreographer`, `traversals` y `DispatchUI`:

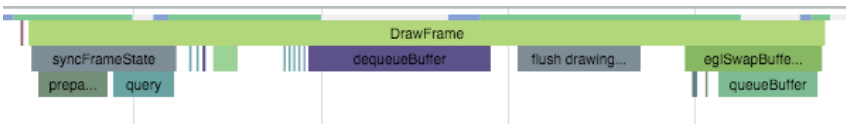


- **JS hilo.** Aquí es donde se ejecuta JavaScript. El nombre del hilo será `mqt_js` o `<...>`dependiendo de qué tan cooperativo sea el kernel en su dispositivo. Para identificarlo si no tiene un nombre, buscar cosas como `JSCall`, `Bridge.executeJSCall`, etc:

- **Native Modules Thread.** Aquí es donde `UIManager` se ejecutan las llamadas al módulo nativo (por ejemplo, el). El nombre del hilo será `mqt_native_modules` o bien o `<...>`. Para identificar que en el último caso, buscar cosas como `NativeCall`, `callJavaModuleMethod` y `onBatchComplete`:

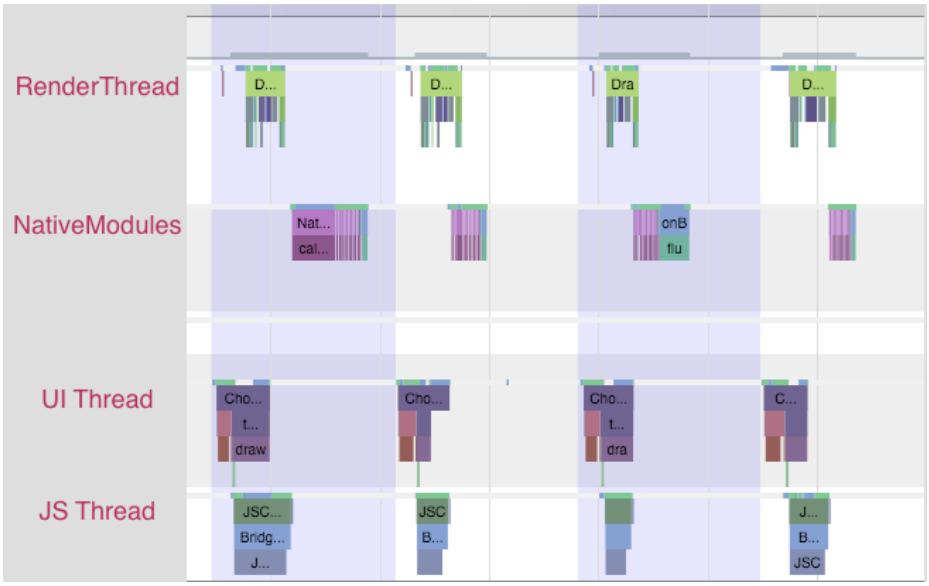


- **Bonificación: Renderizar hilo.** Si está utilizando Android L (5.0) y superior, también tendrá un hilo de renderizado en su aplicación. Este hilo genera los comandos de OpenGL reales utilizados para dibujar su UI. El nombre del hilo será `RenderThread` o bien o `<...>`. Para identificarlo en el último caso, busque cosas como `DrawFrame` y `queueBuffer`:

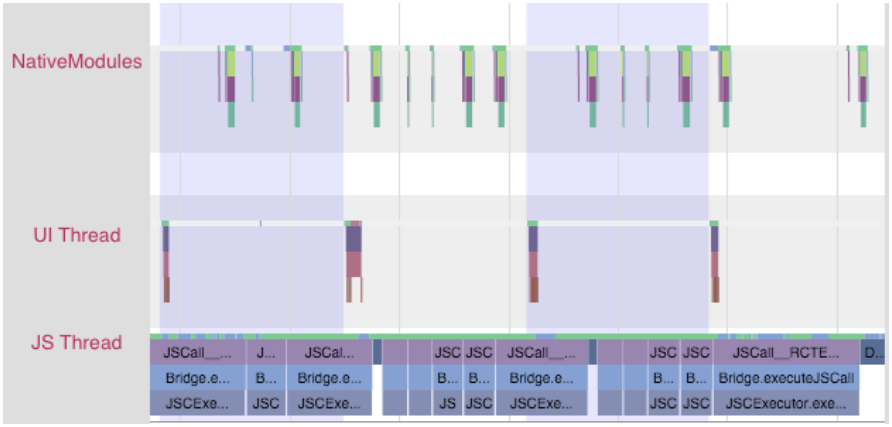


Identificando un culpable

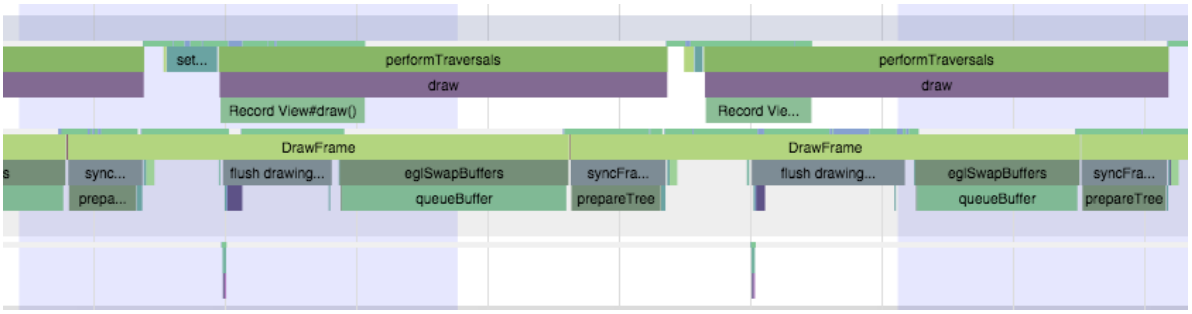
Una animación suave debería verse de la siguiente manera:



Cada cambio de color es un marco: recuerde que para mostrar un marco, todo nuestro trabajo de IU debe realizarse al final de ese período de 16 ms. Observe que ningún hilo está trabajando cerca del límite del marco. Una prestación de aplicaciones como esta se está renderizando a 60 FPS.
Si notas chop, sin embargo, es posible que veas algo como esto:



¡Observe que el hilo JS se está ejecutando básicamente todo el tiempo, y más allá de los límites del marco! Esta aplicación no se está renderizando a 60 FPS. En este caso, **el problema radica en JS** .
También puede ver algo como esto:

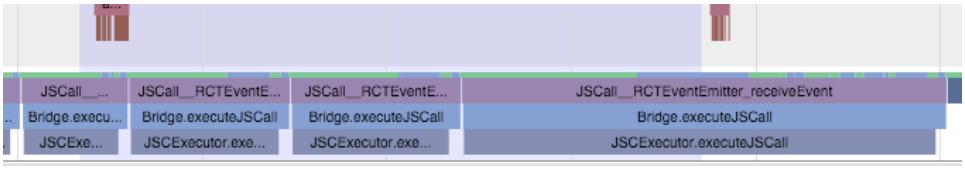


En este caso, la UI y los hilos de representación son los que tienen trabajo cruzando los límites del marco. La interfaz de usuario que intentamos procesar en cada cuadro requiere mucho trabajo. En este caso, **el problema radica en las vistas nativas que se representan** .

En este punto, tendrá información muy útil para informar sus próximos pasos.

Resolviendo problemas de JavaScript

Si identificó un problema de JS, busque pistas en el JS específico que está ejecutando. En el escenario anterior, vemos que `RCTEventEmitter` se lo llama varias veces por fotograma. Aquí hay un acercamiento del subproceso JS del rastreo anterior:



Esto no parece correcto. ¿Por qué se lo llama con tanta frecuencia? ¿Son en realidad diferentes eventos? Las respuestas a estas preguntas probablemente dependerán de su código de producto. Y muchas veces, querrá examinar shouldComponentUpdate .

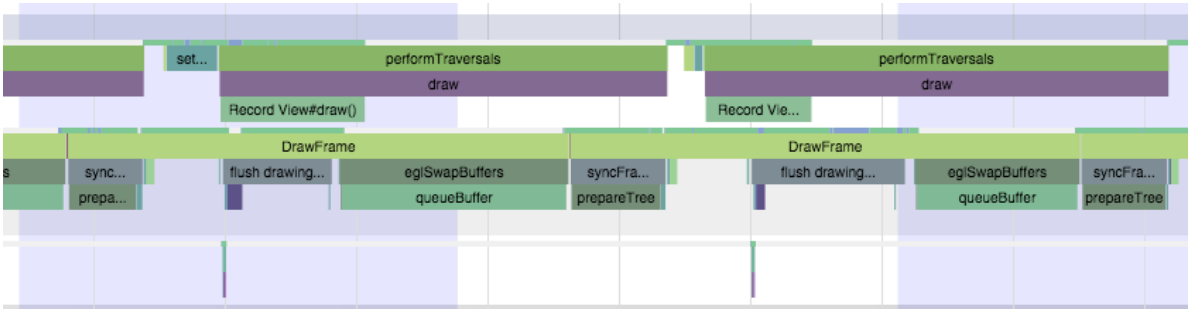
Resolución de problemas de UI nativos

Si identificó un problema de UI nativo, generalmente hay dos escenarios:

1. la interfaz de usuario que está tratando de dibujar en cada cuadro implica mucho trabajo en la GPU, o
2. Está construyendo una nueva interfaz de usuario (UI) durante la animación / interacción (por ejemplo, cargando contenido nuevo durante un desplazamiento).

Demasiado GPU trabajo

En el primer escenario, verá un trazo que tiene el subproceso de interfaz de usuario y / o el hilo de procesamiento con el siguiente aspecto:



Observe que la gran cantidad de tiempo que se pasa `DrawFrame`cruza los límites del marco. Este es el tiempo dedicado a esperar a que la GPU drene su búfer de comando del cuadro anterior.

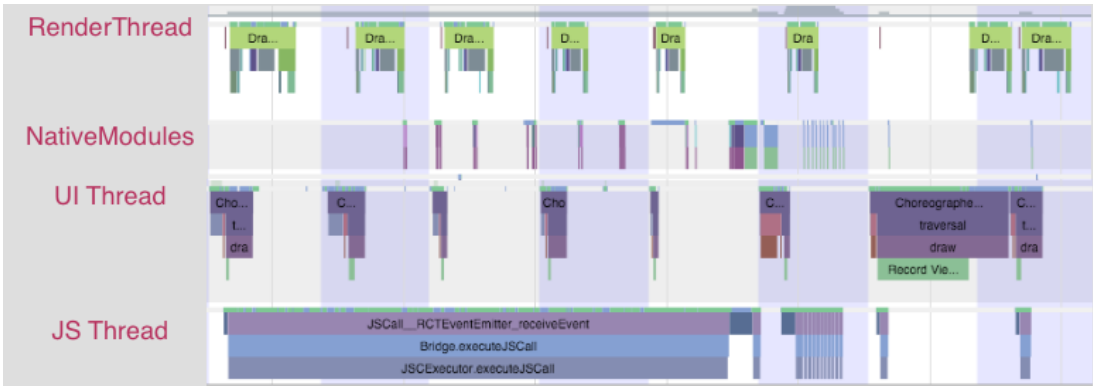
Para mitigar esto, debes:

- Investigue el uso `renderToHardwareTextureAndroid` de contenido complejo y estático que se está animando / transformando (por ejemplo, las `Navigator` animaciones diapositiva / alfa)
- asegúrese de que está **no** está usando `needsOffscreenAlphaCompositing`, que está desactivado por defecto, ya que aumenta en gran medida la carga por trama en la GPU en la mayoría de los casos.

Si esto no ayuda y desea profundizar en lo que realmente hace la GPU, puede consultar Tracer para OpenGL ES .

Creando nuevas vistas en el hilo de la interfaz de usuario

En el segundo escenario, verás algo más como esto:



Tenga en cuenta que primero el subproceso JS piensa un poco, luego ve que se realiza un trabajo en el subproceso de módulos nativos, seguido de un costoso recorrido en el subproceso de interfaz de usuario.
No hay una manera fácil de mitigar esto a menos que pueda posponer la creación de una nueva IU hasta después de la interacción, o puede simplificar la IU que está creando. El equipo nativo de Reaction está trabajando en una solución de nivel de infraestructura para esto que permitirá que se cree y configure una nueva interfaz de usuario fuera del hilo principal, permitiendo que la interacción continúe sin problemas.

Gesture Responder System

El sistema de respuestas gestuales administra el ciclo de vida de los gestos en su aplicación. Un toque puede pasar por varias fases a medida que la aplicación determina cuál es la intención del usuario. Por ejemplo, la aplicación necesita determinar si el toque se desplaza, se desliza en un widget o toca. Esto incluso puede cambiar durante la duración de un toque. También puede haber múltiples toques simultáneos.

El sistema de respuesta táctil es necesario para permitir que los componentes negocien estas interacciones táctiles sin ningún conocimiento adicional sobre sus componentes principales o secundarios. Este sistema está implementado en `ResponderEventPlugin.js`, que contiene más detalles y documentación.

Mejores prácticas

Para que tu aplicación se sienta genial, cada acción debe tener los siguientes atributos:

- Retroalimentación / resaltado: muestra al usuario qué manipula su tacto y qué sucederá cuando lo suelte.
- Capacidad de cancelación: al realizar una acción, el usuario debería poder abortarla al tocarla arrastrando el dedo hacia afuera

Estas características hacen que los usuarios se sientan más cómodos mientras usan una aplicación, ya que les permite a las personas experimentar e interactuar sin temor a cometer errores.

TouchableHighlight y Touchable *

El sistema de respuesta puede ser complicado de usar. Así que hemos proporcionado una `Touchable` implementación abstracta para cosas que deberían ser "tappables". Esto utiliza el sistema de respuesta y le permite configurar fácilmente las interacciones de grifo de forma declarativa. Use en `TouchableHighlight` cualquier lugar donde use un botón o enlace en la web.

Ciclo de vida del respondedor

Una vista puede convertirse en la respuesta táctil mediante la implementación de los métodos de negociación correctos. Hay dos métodos para preguntarle a la vista si quiere convertirse en respondedor:

- `View.props.onStartShouldSetResponder: (evt) => true,` - ¿Esta vista quiere convertirse en respondedor al inicio de un toque?
- `View.props.onMoveShouldSetResponder: (evt) => true,` - Llamado por cada movimiento táctil en la Vista cuando no es el respondedor: ¿esta vista quiere "reclamar" la capacidad de respuesta táctil?

Si la vista devuelve verdadero e intenta convertirse en el respondedor, ocurrirá una de las siguientes:

- `View.props.onResponderGrant: (evt) => {}` - La vista ahora está respondiendo por eventos táctiles. Este es el momento de resaltar y mostrar al usuario lo que está sucediendo
- `View.props.onResponderReject: (evt) => {}` - Algo más es el respondedor en este momento y no lo lanzará

Si la vista está respondiendo, se pueden llamar a los siguientes controladores:

- `View.props.onResponderMove: (evt) => {}` - El usuario está moviendo su dedo
- `View.props.onResponderRelease: (evt) => {}` - Disparado al final del toque, es decir, "touchUp"
- `View.props.onResponderTerminationRequest: (evt) => true` - Algo más quiere convertirse en respondedor. ¿Debería esta vista liberar al respondedor? El regreso verdadero permite la liberación
- `View.props.onResponderTerminate: (evt) => {}` - El respondedor ha sido tomado de la Vista. Puede ser tomado por otras vistas después de una llamada `onResponderTerminationRequest`, o puede ser tomado por el sistema operativo sin preguntar (sucede con el centro de control / centro de notificaciones en iOS)

`evt` es un evento táctil sintético con la siguiente forma:

- `nativeEvent`
  - `changedTouches` - Matriz de todos los eventos táctiles que han cambiado desde el último evento
  - `identifier` - La identificación del tacto
  - `locationX` - La posición X del toque, relativa al elemento
  - `locationY` - La posición Y del toque, relativa al elemento
  - `pageX` - La posición X del toque, relativa al elemento raíz
  - `pageY` - La posición Y del toque, relativa al elemento raíz
  - `target` - La id del nodo del elemento que recibe el evento táctil
  - `timestamp` - Un identificador de tiempo para el tacto, útil para el cálculo de velocidad
  - `touches` - Matriz de todos los toques actuales en la pantalla

Capture ShouldSet Handlers

`OnStartShouldSetResponder` y `onMoveShouldSetResponder` son llamados con un patrón de burbujeo, donde el nodo más profundo se llama primero. Eso significa que el componente más profundo se convertirá en respondedor cuando múltiples vistas devuelvan verdadero para los `*ShouldSetResponder` manejadores. Esto es deseable en la mayoría de los casos, porque asegura que todos los controles y botones sean utilizables.

Sin embargo, a veces un padre querrá asegurarse de que se convierta en respondedor. Esto se puede manejar usando la fase de captura. Antes de que el sistema de respuesta burbujee desde el componente más profundo, hará una fase de captura, disparando `on*ShouldSetResponderCapture`. Entonces, si una Vista padre quiere evitar que el niño se convierta en respondedor en un inicio táctil, debe tener un `onStartShouldSetResponderCapture` controlador que devuelva verdadero.

- `View.props.onStartShouldSetResponderCapture: (evt) => true,`
- `View.props.onMoveShouldSetResponderCapture: (evt) => true,`

PanResponder

Para la interpretación de gestos de alto nivel, echa un vistazo a PanResponder .

Entorno de JavaScript

JavaScript Runtime

Al usar React Native, vas a ejecutar tu código JavaScript en dos entornos:

- En los simuladores y dispositivos iOS, los emuladores y dispositivos Android React Native usan JavaScriptCore, que es el motor de JavaScript que impulsa Safari. En iOS, JSC no usa JIT debido a la ausencia de memoria ejecutable grabable en las aplicaciones de iOS.
- Al usar la depuración de Chrome, ejecuta todo el código JavaScript dentro de Chrome y se comunica con el código nativo a través de WebSocket. Entonces estás usando V8 .

Si bien ambos entornos son muy similares, puede terminar golpeando algunas inconsistencias. Probablemente vamos a experimentar con otros motores JS en el futuro, por lo que es mejor evitar confiar en detalles de cualquier tiempo de ejecución.

Transformadores de sintaxis JavaScript

Los transformadores de sintaxis hacen que el código de escritura sea más agradable al permitirle usar la nueva sintaxis de JavaScript sin tener que esperar el soporte de todos los intérpretes.

A partir de la versión 0.5.0, React Native se envía con el compilador de Babel JavaScript . Consulte la documentación de Babel sobre sus transformaciones compatibles para obtener más detalles.

Aquí hay una lista completa de las transformaciones habilitadas de React Native .

ES5

- Palabras reservadas: `promise.catch(function() { });`

ES6

- Funciones de flecha :`<C onPress={() => this.setState({pressed: true})}`
- Alcance del bloque :`let greeting = 'hi';`
- Propagación de llamada :`Math.max(...array);`
- Clases :`class C extends React.Component { render() { return <View />; } }`
- Constantes :`const answer = 42;`
- Destructuring :`var {isActive, style} = this.props;`
- para ... de :`for (var num of [1, 2, 3]) {}`
- Módulos :`import React, { Component } from 'react';`
- Propiedades calculadas :`var key = 'abc'; var obj = {[key]: 10};`
- Método conciso de objeto :`var obj = { method() { return 10; } };`
- Notación corta de objetos :`var name = 'vjeux'; var obj = { name };`
- Parametros de descanso :`function(type, ...args) { }`
- Literales de la plantilla :`var who = 'world'; var str = `Hello ${who}`;`

ES7

- Extensión de objeto :`var extended = { ...obj, a: 10 };`
- Función Coma final :`function f(a, b, c,) { }`
- Funciones Async : `async function doStuffAsync() { const foo = await doOtherStuffAsync(); };`

Específico

- JSX :`<View style={{color: 'red'}} />`
- Flujo :`function foo(x: ?number): string {}`

Polyfills

Muchas funciones de estándares también están disponibles en todos los tiempos de ejecución soportados de JavaScript.

Navegador

- consola. {log, warn, error, info, trace, table}
- CommonJS requieren
- XMLHttpRequest, buscar
- {set, clear} {Timeout, Interval, Inmediate}, {solicitud, cancelación} AnimationFrame
- navigator.geolocation

ES6

- Object.assign
- String.prototype. { StartsWith , endsWith , repeat , includes }
- Array.from
- Array.prototype. { Encontrar , FindIndex , incluye }

ES7

- Objeto. { Entradas , valores }

Específico

- `__DEV__`

Manipulación directa

A veces es necesario realizar cambios directamente en un componente sin utilizar state / props para desencadenar una nueva representación de todo el subárbol. Al utilizar React en el navegador, por ejemplo, a veces es necesario modificar directamente un nodo DOM, y lo mismo es cierto para las vistas en las aplicaciones móviles. `SetNativeProps` es el React Native equivalente a establecer propiedades directamente en un nodo DOM.

> Use setNativeProps cuando la repetición frecuente crea un cuello de botella de rendimiento
>
> La manipulación directa no será una herramienta a la que recurras frecuentemente; por lo general, solo lo utilizará para crear animaciones continuas para evitar la sobrecarga de representar la jerarquía de componentes y reconciliar muchas vistas. `SetNativeProps` es imperativo y almacena el estado en la capa nativa (DOM, UIView, etc.) y no dentro de los componentes de React, lo que hace que sea más difícil razonar sobre su código. Antes de usarlo, intente resolver su problema con `setState` y shouldComponentUpdate .

setNativeProps con TouchableOpacity

TouchableOpacity utiliza `setNativeProps` internamente para actualizar la opacidad de su componente hijo:

```
setOpacityTo(value) {
        // Redacted: animation related code
        this.refs[CHILD_REF].setNativeProps({
                opacity: value
        });
},
```

Esto nos permite escribir el siguiente código y saber que el hijo tendrá su opacidad actualizada en respuesta a los grifos, sin que el hijo tenga ningún conocimiento de ese hecho ni requiera ningún cambio en su implementación:

```
<TouchableOpacity onPress={this._handlePress}>
        <View style={styles.button}>
                <Text>Press me!</Text>
        </View>
</TouchableOpacity>
```

Imaginemos que `setNativeProps` no estaba disponible. Una forma de implementarlo con esa restricción es almacenar el valor de opacidad en el estado y luego actualizar ese valor cada vez que `onPress`se active:

```
constructor(props) {
        super(props);
        this.state = { myButtonOpacity: 1, };
}

render() {
        return (
                <TouchableOpacity onPress={() => this.setState({myButtonOpacity: 0.5})}
                                  onPressOut={() => this.setState({myButtonOpacity: 1})}>
                        <View style={[styles.button, {opacity: this.state.myButtonOpacity}]}>
                                <Text>Press me!</Text>
                        </View>
                </TouchableOpacity>
        )
}
```

Esto es computacionalmente intensivo en comparación con el ejemplo original: React necesita volver a representar la jerarquía de componentes cada vez que cambia la opacidad, a pesar de que otras propiedades de la vista y sus elementos secundarios no han cambiado. Por lo general, esta sobrecarga no es una preocupación, pero al realizar animaciones continuas y responder a los gestos, la optimización juiciosa de los componentes puede mejorar la fidelidad de sus animaciones.

Si nos fijamos en la implementación de `setNativeProps` en NativeMethodsMixin.js , notará que se trata de una envoltura alrededor `RCTUIManager.updateView` (esta es exactamente la misma llamada de función que resulta de volver a renderizar), vea receiveComponent en ReactNativeBaseComponent.js .

Componentes compuestos y setNativeProps

Los componentes compuestos no están respaldados por una vista nativa, por lo que no puede invocarlos `setNativeProps`. Considera este ejemplo:

```
import React from 'react';
import { Text, TouchableOpacity, View } from 'react-native';

class MyButton extends React.Component {
  render() {
    return (
      <View>
        <Text>{this.props.label}</Text>
      </View>
    )
  }
}

export default class App extends React.Component {
  render() {
    return (
      <TouchableOpacity>
        <MyButton label="Press me!" />
      </TouchableOpacity>
    )
  }
}
```

Si ejecuta este inmediatamente se ve este error: `Touchable child must either be native or forward setNativeProps to a native component`. Esto ocurre porque `MyButton` no está respaldado directamente por una vista nativa cuya opacidad debe establecerse. Puede pensarlo de esta manera: si define un componente con el `createReactClass` que no esperaría poder establecer un estilo prop y hacer ese trabajo, tendría que pasar el apuntalamiento de estilo a un niño, a menos que esté envolviendo un componente nativo. Del mismo modo, vamos a reenviar `setNativeProps` a un componente hijo con respaldo nativo.

Adelante setNativeProps a un hijo

Todo lo que tenemos que hacer es proporcionar un `setNativeProps` método en nuestro componente que llame `setNativeProps` al hijo apropiado con los argumentos dados.

```
import React from 'react';
import { Text, TouchableOpacity, View } from 'react-native';

class MyButton extends React.Component {
  setNativeProps = (nativeProps) => {
    this._root.setNativeProps(nativeProps);
  }

  render() {
    return (
      <View ref={component => this._root = component} {...this.props}>
        <Text>{this.props.label}</Text>
      </View>
    )
  }
}

export default class App extends React.Component {
  render() {
    return (
      <TouchableOpacity>
        <MyButton label="Press me!" />
      </TouchableOpacity>
    )
  }
}
```

¡Ahora puedes usar `MyButton` dentro de `TouchableOpacity` ! Una nota al margen para mayor claridad: utilizamos la sintaxis de devolución de llamada refaquí, en lugar de la ref tradicional basada en cadena.
Es posible que haya notado que pasamos todos los puntales a la vista secundaria utilizando `{...this.props}`. La razón de esto es que en`TouchableOpacity` realidad es un componente compuesto, por lo que, además de depender `setNativeProps` de su hijo, también requiere que el niño realice el manejo táctil. Para hacer esto, pasa varios accesorios que vuelven a llamar al `TouchableOpacity` componente. `TouchableHighlight`, en cambio, está respaldado por una vista nativa y solo requiere que lo implementemos `setNativeProps`.

setNativeProps para borrar el valor de TextInput

Otro caso de uso muy común `setNativeProps`es borrar el valor de un TextInput. El `controlled` prop de TextInput a veces puede soltar caracteres cuando el número `bufferDelay` es bajo y el usuario escribe muy rápido. Algunos desarrolladores prefieren omitir este accesorio por completo y en su lugar utilizar `setNativeProps` para

manipular directamente el valor de TextInput cuando sea necesario. Por ejemplo, el siguiente código demuestra borrar la entrada cuando toca un botón:

```
import React from 'react';
import { TextInput, Text, TouchableOpacity, View } from 'react-native';

export default class App extends React.Component {
  clearText = () => {
    this._textInput.setNativeProps({text: ''});
  }

  render() {
    return (
      <View style={{flex: 1}}>
        <TextInput
          ref={component => this._textInput = component}
          style={{height: 50, flex: 1, marginHorizontal: 20, borderWidth: 1, borderColor:
'#ccc'}}
        />
        <TouchableOpacity onPress={this.clearText}>
          <Text>Clear text</Text>
        </TouchableOpacity>
      </View>
    );
  }
}
```

Evitando conflictos con la función de renderización

Si actualiza una propiedad que también es administrada por la función de renderizado, podría terminar con algunos errores impredecibles y confusos porque cada vez que el componente se renueva y esa propiedad cambia, cualquier valor que se haya establecido previamente `setNativeProps` se ignorará por completo y se anulará.

setNativeProps y shouldComponentUpdate

Mediante la aplicación inteligente `shouldComponentUpdate` puede evitar la sobrecarga innecesaria involucrada en la reconciliación de subárboles de componentes sin cambios, hasta el punto en que puede ser lo suficientemente eficiente para usar en `setState` lugar de hacerlo `setNativeProps`.

Otros métodos nativos

Los métodos descritos aquí están disponibles en la mayoría de los componentes predeterminados proporcionados por React Native. Tenga en cuenta, sin embargo, que son *no* disponibles en los componentes compuestos que no

están respaldados directamente por un punto de vista nativo. Esto generalmente incluirá la mayoría de los componentes que defina en su propia aplicación.

medida (devolución de llamada)

Determina la ubicación en la pantalla, el ancho y el alto de la vista determinada y devuelve los valores a través de una devolución de llamada asíncrona. Si tiene éxito, se llamará a la devolución de llamada con los siguientes argumentos:

- X
- y
- anchura
- altura
- páginaX
- pageY

Tenga en cuenta que estas mediciones no están disponibles hasta después de que la representación se haya completado en nativo. Si necesita las medidas lo antes posible, considere usar el `onLayout`puntal en su lugar.

measureInWindow (devolución de llamada)

Determina la ubicación de la vista dada en la ventana y devuelve los valores mediante una devolución de llamada asíncrona. Si la vista raíz Reaccionar está incrustada en otra vista nativa, esto le dará las coordenadas absolutas. Si tiene éxito, se llamará a la devolución de llamada con los siguientes argumentos:

- X
- y
- anchura
- altura

measureLayout (relativeToNativeNode, onSuccess, onFail)

Me gusta `measure()`, pero mide la vista relativa de un antepasado, especificado como `relativeToNativeNode`. Esto significa que las x, y devueltas son relativas al origen x, y de la vista del antepasado.

Como siempre, para obtener un identificador de nodo nativo para un componente, puede usar `ReactNative.findNodeHandle(component)`.

focus ()

Pide enfoque para la entrada o vista dada. El comportamiento exacto desencadenado dependerá de la plataforma y el tipo de vista.

blur ()

Elimina el foco de una entrada o vista. Esto es lo opuesto a `focus()`.

Color Reference

Los componentes en React Native se diseñan usando JavaScript . Las propiedades de color generalmente coinciden con el funcionamiento de CSS en la web .

Rojo-verde-azul

React Native admite `rgb()` y `rgba()` en notación tanto hexadecimal como funcional:

- `'#f0f'` (#rgb)
- `'#ff00ff'` (#rrggbb)
- `'rgb(255, 0, 255)'`
- `'rgba(255, 255, 255, 1.0)'`
- `'#f0ff'` (#rgba)
- `'#ff00ff00'` (#rrggbbaa)

Hue-saturation-lightness

`hsl()` y `hsla()` es compatible con la notación funcional:

- `'hsl(360, 100%, 100%)'`
- `'hsla(360, 100%, 100%, 1.0)'`

`transparent`

Este es un atajo para `rgba(0,0,0,0)`:

- `'transparent'`

Colores nombrados

También puede usar nombres de colores como valores. React Native sigue la especificación CSS3 :

- aliceblue (# f0f8ff)
- antiquewhite (# faebd7)
- aqua (# 00ffff)
- aguamarina (# 7fffd4)
- azul (# f0ffff)
- beige (# f5f5dc)
- bisque (# ffe4c4)
- negro (# 000000)
- blanchedalmond (#ffebcd)
- azul (# 0000ff)
- blueviolet (# 8a2be2)
- marrón (# a52a2a)
- burlywood (# deb887)
- cadetblue (# 5f9ea0)
- chartreuse (# 7fff00)
- chocolate (# d2691e)
- coral (# ff7f50)
- cornflowerblue (# 6495ed)
- cornsilk (# fff8dc)
- carmesí (# dc143c)

- cian (# 00ffff)
- azul oscuro (# 00008b)
- darkcyan (# 008b8b)
- darkgoldenrod (# b8860b)
- darkgray (# a9a9a9)
- verde oscuro (# 006400)
- darkgrey (# a9a9a9)
- darkkhaki (# bdb76b)
- Darkmagenta (# 8b008b)
- darkolivegreen (# 556b2f)
- darkorange (# ff8c00)
- Darkorchid (# 9932cc)
- darkred (# 8b0000)
- darksalmon (# e9967a)
- darkseagreen (# 8fbc8f)
- darkslateblue (# 483d8b)
- Darkslategrey (# 2f4f4f)
- darkturquoise (# 00ced1)
- Darkviolet (# 9400d3)
- deeppink (# ff1493)
- deepskyblue (# 00bfff)
- dimgray (# 696969)
- dimgrey (# 696969)
- dodgerblue (# 1e90ff)
- ladrillo refractario (# b22222)
- floralwhite (# fffaf0)
- forestgreen (# 228b22)
- fucsia (# ff00ff)
- winsboro (#cdcdc)
- ghostwhite (# f8f8ff)
- oro (# ffd700)
- vara de oro (# daa520)
- gris (# 808080)
- verde (# 008000)
- greenyellow (# adff2f)
- gris (# 808080)
- mielada (# f0fff0)
- hotpink (# ff69b4)
- indianred (# cd5c5c)
- índigo (# 4b0082)
- marfil (# fffff0)
- caqui (# f0e68c)
- lavanda (# e6e6fa)
- Lavenderblush (# fff0f5)
- lawngreen (# 7cfc00)
- lemonchiffon (#fffacd)
- azul claro (# add8e6)

- lightcoral (# f08080)
- Lightcyan (# e0ffff)
- lightgoldenrodyellow (# fafad2)
- lightgray (# d3d3d3)
- verde claro (# 90ee90)
- lightgrey (# d3d3d3)
- lightpink (# ffb6c1)
- Lightsalmon (# ffa07a)
- lightseagreen (# 20b2aa)
- lightskyblue (# 87cefa)
- lightslategrey (# 778899)
- lightsteelblue (# b0c4de)
- lightyellow (# ffffe0)
- lima (# 00ff00)
- limegreen (# 32cd32)
- lino (# faf0e6)
- magenta (# ff00ff)
- granate (# 800000)
- mediumaquamarine (# 66cdaa)
- mediumblue (# 0000cd)
- mediumorchid (# ba55d3)
- mediumpurple (# 9370db)
- mediumseagreen (# 3cb371)
- mediumslateblue (# 7b68ee)
- verde mediano (# 00fa9a)
- medioturquesa (# 48d1cc)
- mediumvioletred (# c71585)
- midnightblue (# 191970)
- Mentacream (# f5fffa)
- mistyrose (# ffe4e1)
- mocasín (# ffe4b5)
- navajowhite (#ffdead)
- navy (# 000080)
- oldlace (# fdf5e6)
- aceituna (# 808000)
- olivedrab (# 6b8e23)
- naranja (# ffa500)
- orangered (# ff4500)
- orquídea (# da70d6)
- palegoldenrod (# eee8aa)
- palegreen (# 98fb98)
- paleturquoise (#afeeee)
- palevioletred (# db7093)
- papayawhip (# ffefd5)
- peachpuff (# ffdab9)
- perú (# cd853f)
- rosa (# ffc0cb)

- ciruela (# dda0dd)
- powderblue (# b0e0e6)
- violeta (# 800080)
- rebeccapurple (# 663399)
- rojo (# ff0000)
- rosybrown (# bc8f8f)
- royalblue (# 4169e1)
- saddlebrown (# 8b4513)
- salmón (# fa8072)
- sandybrown (# f4a460)
- seagreen (# 2e8b57)
- concha (# fff5ee)
- sienna (# a0522d)
- plateado (# c0c0c0)
- skyblue (# 87ceeb)
- slateblue (# 6a5acd)
- slategray (# 708090)
- nieve (#fffafa)
- springgreen (# 00ff7f)
- steelblue (# 4682b4)
- tan (# d2b48c)
- verde azulado (# 008080)
- cardo (# d8bfd8)
- tomate (# ff6347)
- turquesa (# 40e0d0)
- violeta (# ee82ee)
- trigo (# f5deb3)
- blanco (#ffffff)
- whitesmoke (# f5f5f5)
- amarillo (# ffff00)
- yellowgreen (# 9acd32)

Integración con aplicaciones existentes

Proyecto con código nativo requerido

Esta página solo se aplica a proyectos realizados con `react-native init` o creados con la aplicación Create React Native que se han expulsado. Para obtener más información sobre la expulsión, consulte la guía en el repositorio Crear repositorio nativo de la aplicación.

React Native es genial cuando estás comenzando una nueva aplicación móvil desde cero. Sin embargo, también funciona bien para agregar una sola vista o flujo de usuario a las aplicaciones nativas existentes. Con unos pocos pasos, puede agregar nuevas funciones basadas en React Native, pantallas, vistas, etc.

Los pasos específicos son diferentes según la plataforma a la que se dirige.

- iOS

Conceptos clave

Las claves para integrar los componentes de React Native en su aplicación iOS son:

1. Configurar las dependencias y la estructura del directorio de React Native.
2. Comprenda qué componentes de React Native usará en su aplicación.
3. Agregue estos componentes como dependencias usando CocoaPods.
4. Desarrolle sus componentes React Native en JavaScript.
5. Agregue una `RCTRootView` a su aplicación de iOS. Esta vista servirá como el contenedor para su componente React Native.
6. Inicie el servidor React Native y ejecute su aplicación nativa.
7. Verifique que el aspecto React Native de su aplicación funciona como se espera.

Requisitos previos

Siga las instrucciones para crear aplicaciones con código nativo en la guía de Inicio para configurar su entorno de desarrollo para compilar aplicaciones Reaccionar nativas para iOS.

1. Configurar la estructura del directorio

Para garantizar una experiencia fluida, cree una nueva carpeta para su proyecto integrado React Native, luego copie su proyecto iOS existente en una `/ios` subcarpeta.

2. Instalar dependencias de JavaScript

Vaya al directorio raíz de su proyecto y cree un nuevo `package.json` archivo con los siguientes contenidos:

```
{
        "name": "MyReactNativeApp",
        "version": "0.0.1",
        "private": true,
        "scripts": {
                "start": "node node_modules/react-native/local-cli/cli.js start"
        }
}
```

A continuación, instalará los paquetes `react` y `react-native`. Abra una terminal o símbolo del sistema, luego navegue hasta el directorio raíz de su proyecto y escriba los siguientes comandos:

```
$ npm install --save react@16.0.0-beta.5 react-native
```

Asegúrese de utilizar la misma versión de React como se especifica en el archivo React Native`package.json` . Esto solo será necesario siempre que React Native dependa de una versión preliminar de React.

Esto creará una nueva carpeta `/node_modules` en el directorio raíz de su proyecto. Esta carpeta almacena todas las dependencias de JavaScript requeridas para construir su proyecto.

3. Instalar CocoaPods

CocoaPods es una herramienta de gestión de paquetes para iOS y desarrollo de macOS. Lo usamos para agregar localmente el código real de React framework en su proyecto actual.

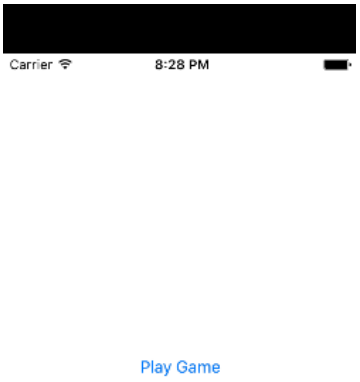
Recomendamos instalar CocoaPods usando Homebrew .

```
$ brew install cocoapods
```

Es técnicamente posible no utilizar CocoaPods, pero eso requeriría adiciones manuales de bibliotecas y enlazadores que complicarían demasiado este proceso.

Agregar React Native a su aplicación

Asuma que la aplicación para integración es un juego de 2048 . Aquí está el aspecto del menú principal de la aplicación nativa sin React Native.



Configurando las dependencias de CocoaPods

Antes de integrar React Native en su aplicación, querrá decidir qué partes del marco Reaccionar Nativo desea integrar. Utilizaremos CocoaPods para especificar en cuál de estas "subespecies" dependerá su aplicación.

La lista de `subspec`s compatibles está disponible en `/node_modules/react-native/React.podspec`.

Generalmente son nombrados por la funcionalidad. Por ejemplo, generalmente siempre querrás el `Core subspec`.

Eso le dará el `AppRegistry`, `StyleSheet`, `View` y el otro núcleo Reaccionar bibliotecas nativas. Si desea agregar la `Text`biblioteca React Native (por ejemplo, para `<Text>`elementos), necesitará la `RCTText subspec`. Si desea la `Image`biblioteca (por ejemplo, para `<Image>`elementos), necesitará la `RCTImage subspec`.

Puedes especificar de qué `subspec` fuente dependerá tu aplicación en un `Podfile` archivo. La forma más fácil de crear una `Podfile` es ejecutando el `init` comando CocoaPods en la `/ios` subcarpeta de su proyecto:

```
$ pod init
```

El `Podfile` contendrá una configuración repetitiva que modificará para su integración. Al final, `Podfile` debería verse algo similar a esto:

```
# El nombre de destino es probablemente el nombre de tu proyecto.
target 'NumberTileGame' do

        # Su 'node_modules' directorio está, probablemente, en la raíz de su proyecto,
        # pero si no , ajustar el `:path` en consecuencia
        pod 'React', :path => '../node_modules/react-native', :subspecs => [
                'Core',
                'DevSupport', # Include this to enable In-App Devmenu if RN >= 0.43
                'RCTText',
                'RCTNetwork',
                'RCTWebSocket', # needed for debugging # Add any other subspecs you want to use in your project
        ]
        # Incluya explícitamente Yoga si está usando RN > = 0.42 . 0
        pod 'yoga', :path => '../node_modules/react-native/ReactCommon/yoga'
end
```

Después de haber creado su `Podfile`, está listo para instalar el pod React Native.

```
$ pod install
```

Debería ver resultados como:

```
Analyzing dependencies
Fetching podspec for `React` from `../node_modules/react-native`
Downloading dependencies
Installing React (0.26.0)
Generating Pods project
Integrating client project
Sending stats
Pod installation complete!
There are 3 dependencies from the Podfile and 1 total
pod installed.
```

Integración de código

Ahora realmente modificaremos la aplicación nativa de iOS para integrar React Native. Para nuestra aplicación de muestra 2048, agregaremos una pantalla de "Puntuación más alta" en React Native.

El componente React Native

El primer fragmento de código que escribiremos es el código Reaccionar nativo real para la nueva pantalla "Puntuación alta" que se integrará en nuestra aplicación.

1. Crea un `index.js`archivo

Primero, crea un `index.js` archivo vacío en la raíz de tu proyecto React Native.

`index.js` es el punto de partida para las aplicaciones React Native, y siempre es necesario. Puede ser un archivo pequeño que sea `require` otro archivo que sea parte de su componente o aplicación React Native, o puede contener todo el código que se necesita para él. En nuestro caso, nos limitaremos a poner todo en `index.js`.

2. Agregue su código de React Native

En tu `index.js`, crea tu componente. En nuestra muestra aquí, agregaremos un `<Text>`componente simple dentro de un estilo`<View>`

```
'use strict';

import React from 'react';
import {
        AppRegistry,
        StyleSheet,
        Text, View
} from 'react-native';

class RNHighScores extends React.Component {
        render() {
                var contents = this.props["scores"].map(
                        score => <Text key={score.name}>{score.name}:{score.value}{"\n"}</Text>
                );
        return (
                <View style={styles.container}>
                        <Text style={styles.highScoresTitle}> 2048 High Scores! </Text>
                        <Text style={styles.scores}> {contents} </Text>
                </View>
                );
        }
}

const styles = StyleSheet.create({
        container: {
                flex: 1,
                justifyContent: 'center',
                alignItems: 'center',
                backgroundColor: '#FFFFFF',
        },
        highScoresTitle: {
                fontSize: 20,
                textAlign: 'center',
                margin: 10,
        },
        scores: {
                textAlign: 'center',
                color: '#333333',
                marginBottom: 5,
        },
});

// Module name
AppRegistry.registerComponent('MyReactNativeApp', () => RNHighScores);
```

`RNHighScores` es el nombre de su módulo que se usará cuando agregue una vista a React Native desde su aplicación iOS.

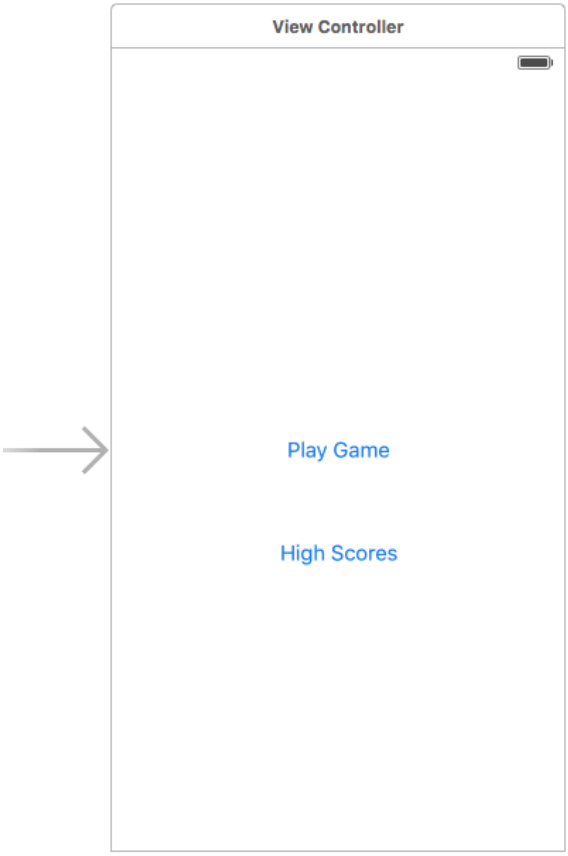
La magia: `RCTRootView`

Ahora que su componente React Native se crea a través de `index.js`, necesita agregar ese componente a uno nuevo o existente `ViewController`. La ruta más fácil de tomar es opcionalmente crear una ruta de evento para su componente y luego agregar ese componente a un existente `ViewController`.

Ataremos nuestro componente React Native con una nueva vista nativa en el `ViewController` que en realidad lo alojará `RCTRootView`.

1. Crea una Ruta de Evento

Puede agregar un nuevo enlace en el menú principal del juego para ir a la página "Reacción máxima" React Native.



2. Controlador de eventos

Ahora agregaremos un controlador de eventos desde el enlace del menú. Se agregará un método al principal `ViewController` de su aplicación. Aquí es donde `RCTRootView` entra en juego.

Cuando construyes una aplicación Reaccionar nativa, usas el paquete Recopilador nativo para crear una `index.bundle`que servirá el servidor React Native. Dentro `index.bundle` estará nuestro `RNHighScore` módulo. Por lo tanto, debemos señalar nuestra `RCTRootView` ubicación del `index.bundle` recurso (vía `NSURL`) y vincularlo al módulo.

Para fines de depuración, iniciaremos sesión que el controlador de eventos fue invocado. Luego, crearemos una cadena con la ubicación de nuestro código Reaccionar nativo que existe dentro de `index.bundle`. Finalmente, crearemos el principal `RCTRootView`. Observe cómo proporcionamos `RNHighScores` el `moduleName` que creamos anteriormente al escribir el código para nuestro componente React Native.

Primero `import` el `RCTRootView` encabezado.

```
#import <React/RCTRootView.h>
```

> El `initialProperties`está aquí con fines de ilustración, así que tenemos algunos datos para nuestra pantalla de alta puntuación. En nuestro componente React Native, lo usaremos `this.props`para obtener acceso a esa información.

En Objective-C:

```
- (IBAction)highScoreButtonPressed:(id)sender {
        NSLog(@"High Score Button Pressed");
        NSURL *jsCodeLocation = [NSURL URLWithString:@"http://localhost:8081/index.bundle?platform=ios"];
        RCTRootView *rootView =
                [[RCTRootView alloc] initWithBundleURL: jsCodeLocation
                        moduleName: @"RNHighScores"
                        initialProperties:
                                @{
                                        @"scores" : @[
                                                @{
                                                        @"name" : @"Alex",
                                                        @"value": @"42"
                                                },
                                                @{
                                                        @"name" : @"Joel",
                                                        @"value": @"10"
                                                }
                                        ]
                                }
                                launchOptions: nil
                ];
        UIViewController *vc = [[UIViewController alloc] init];
        vc.view = rootView;
        [self presentViewController:vc animated:YES completion:nil];
}
```

En Swift:

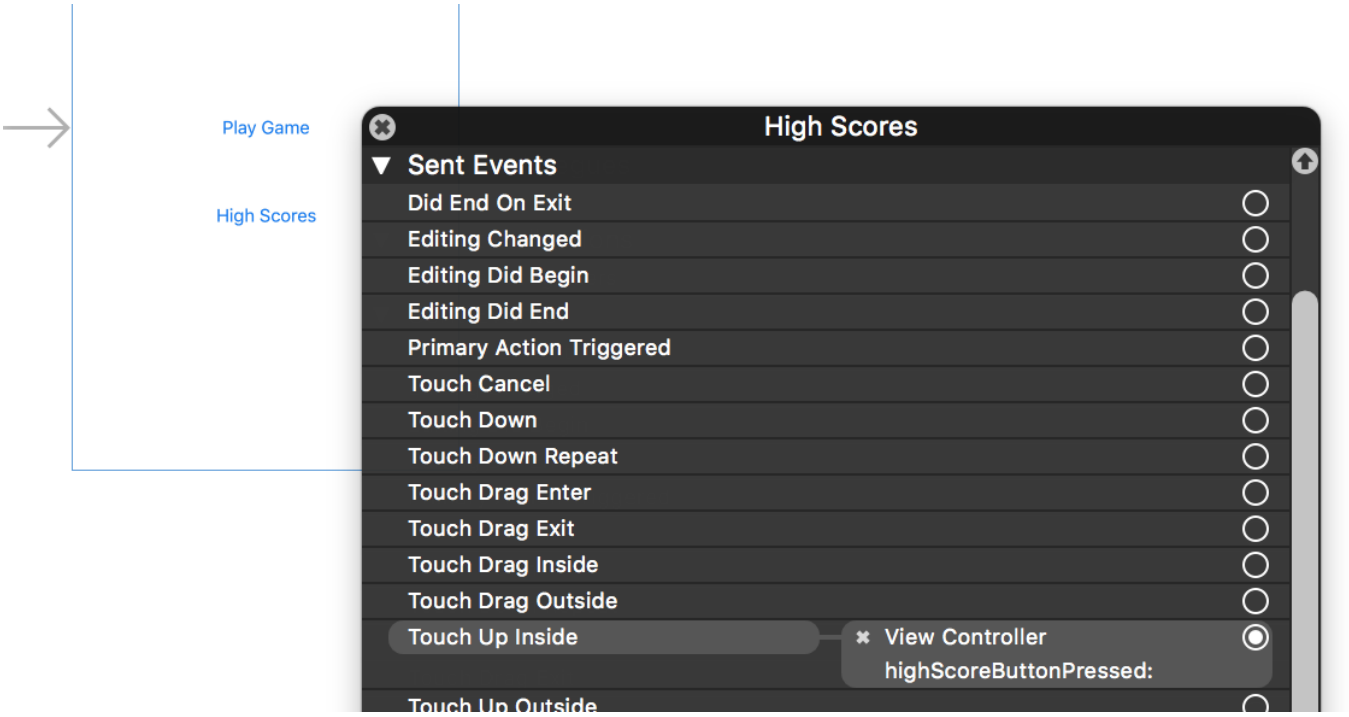
```
@IBAction func highScoreButtonTapped(sender : UIButton) {
        NSLog("Hello")
        let jsCodeLocation = URL(string: "http://localhost:8081/index.bundle?platform=ios")
        let mockData:NSDictionary = ["scores":
                [
                        ["name":"Alex", "value":"42"],
                        ["name":"Joel", "value":"10"]
                ]
        ]
        let rootView = RCTRootView(
                bundleURL: jsCodeLocation,
                moduleName: "RNHighScores",
                initialProperties: mockData as [NSObject : AnyObject],
                launchOptions: nil
        )
        let vc = UIViewController()
        vc.view = rootView
        self.present(vc, animated: true, completion: nil)
}
```

Tenga en cuenta que se `RCTRootView initWithURL` inicia una nueva VM de JSC. Para ahorrar recursos y simplificar la comunicación entre las vistas de RN en diferentes partes de su aplicación nativa, puede tener múltiples vistas con la tecnología de

React Native asociadas con un solo tiempo de ejecución JS. Para hacer eso, en lugar de usar `[RCTRootView alloc] initWithURL`, use `RCTBridge initWithBundleURL`para crear un puente y luego usar `RCTRootView initWithBridge`. Cuando mueva su aplicación a producción, `NSURL` puede apuntar a un archivo prearmado en el disco a través de algo así como `[[NSBundle mainBundle] URLForResource:@"main" withExtension:@"jsbundle"];`. Puede usar la `react-native-xcode.sh` secuencia de comandos `node_modules/react-native/scripts/`para generar ese archivo pre-agrupado.

3. Wire Up

Conecte el nuevo enlace en el menú principal al método del controlador de eventos recién agregado.



Una de las formas más sencillas de hacerlo es abrir la vista en el guión gráfico y hacer clic derecho en el nuevo enlace. Seleccione algo como el `Touch Up Inside` evento, arrástrelo al guión gráfico y luego seleccione el método creado de la lista proporcionada.

Pon a prueba tu integración

Ya ha realizado todos los pasos básicos para integrar React Native con su aplicación actual. Ahora comenzaremos el `index.bundle` paquete React Native para compilar el paquete y el servidor que se ejecutará `localhost` para servirlo.

1. Agregue la excepción de seguridad de transporte de aplicaciones

Apple ha bloqueado la carga implícita de recursos HTTP transparentes. Entonces, necesitamos agregar el siguiente `Info.plist` archivo de nuestro proyecto (o equivalente).

```
<key>NSAppTransportSecurity</key>
<dict>
        <key>NSExceptionDomains</key>
        <dict>
                <key>localhost</key>
                <dict>
                        <key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
                        <true/>
                </dict>
        </dict>
</dict>
```

App Transport Security es bueno para sus usuarios. Asegúrese de volver a habilitarlo antes de lanzar su aplicación para la producción.

2. Ejecute el paquete

Para ejecutar su aplicación, primero debe iniciar el servidor de desarrollo. Para hacer esto, simplemente ejecute el siguiente comando en el directorio raíz de su proyecto React Native:

```
$ npm start
```

3. Ejecute la aplicación

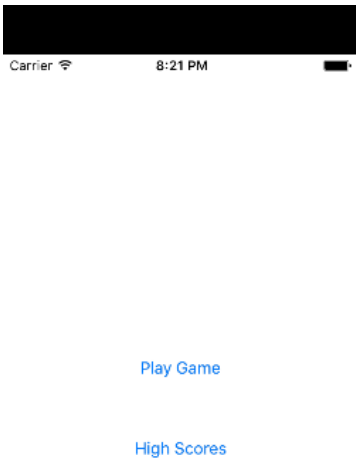
Si está utilizando Xcode o su editor favorito, cree y ejecute su aplicación iOS nativa de forma normal.

Alternativamente, puede ejecutar la aplicación desde la línea de comando usando:

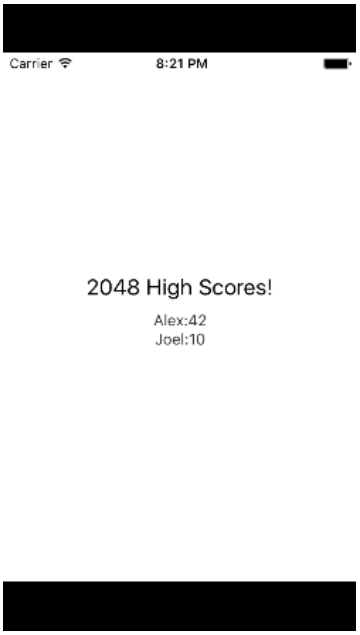
```
# Desde la raíz de tu proyecto
$ react-native run-ios
```

En nuestra aplicación de ejemplo, debería ver el enlace a "Puntajes altos" y luego, cuando haga clic en él, verá la representación de su componente React Native.

Aquí está la pantalla de inicio de la aplicación *nativa* :



Aquí está la pantalla de puntuación *más* alta de *React Native* :



Si tiene problemas de resolución del módulo al ejecutar su aplicación, consulte este problema de GitHub para obtener información y una posible resolución. Este comentario parece ser la última resolución posible.

Ver el código

Puede examinar el código que agregó la pantalla React Native a nuestra aplicación de muestra en GitHub .

¿Ahora que?

En este punto, puedes continuar desarrollando tu aplicación como siempre. Consulte nuestros documentos de depuración e implementación para obtener más información sobre cómo trabajar con React Native.

Android (Java)

Conceptos clave

Las claves para integrar los componentes de React Native en su aplicación Android son:

1. Configurar las dependencias y la estructura del directorio de React Native.
2. Desarrolle sus componentes React Native en JavaScript.
3. Agregue una `ReactRootView` a su aplicación de Android. Esta vista servirá como el contenedor para su componente React Native.
4. Inicie el servidor React Native y ejecute su aplicación nativa.
5. Verifique que el aspecto React Native de su aplicación funciona como se espera.

Requisitos previos

Siga las instrucciones para crear aplicaciones con código nativo en la guía de Inicio para configurar su entorno de desarrollo para compilar aplicaciones Reaccionar nativas para Android.

1. Configurar la estructura del directorio

Para garantizar una experiencia fluida, cree una nueva carpeta para su proyecto integrado React Native, luego copie su proyecto Android existente en una `/android` subcarpeta.

2. Instalar dependencias de JavaScript

Vaya al directorio raíz de su proyecto y cree un nuevo `package.json` archivo con los siguientes contenidos:

```
{
        "name": "MyReactNativeApp",
        "version": "0.0.1",
        "private": true,
        "scripts": {
                "start": "node node_modules/react-native/local-cli/cli.js start"
        }
}
```

A continuación, instalará los paquetes `react` y `react-native`. Abra una terminal o símbolo del sistema, luego navegue hasta el directorio raíz de su proyecto y escriba los siguientes comandos:

```
$ npm install --save react@16.0.0-beta.5 react-native
```

> Asegúrese de utilizar la misma versión de React como se especifica en el archivo React Native`package.json` . Esto solo será necesario siempre que React Native dependa de una versión preliminar de React.

Esto creará una nueva `/node_modules`carpeta en el directorio raíz de su proyecto. Esta carpeta almacena todas las dependencias de JavaScript requeridas para construir su proyecto.

Agregar React Native a su aplicación

Configurando maven

Agregue la dependencia React Native al `build.gradle` archivo de su aplicación :

```
dependencies {
        ... compile "com.facebook.react:react-native:+"
        // From node_modules.
}
```

> Si quiere asegurarse de que siempre está utilizando una versión específica de React Native en su versión original, reemplace `+`con una versión real de React Native que haya descargado `npm`.

Agregue una entrada para el directorio local de React Native maven `build.gradle`. Asegúrese de agregarlo al bloque "allprojects":

```
allprojects {
        repositories {
                ...
                maven {
                        // All of React Native (JS, Android binaries) is installed from npm
                        url "$rootDir/node_modules/react-native/android"
                }
        }
... }
```

> ¡Asegúrate de que la ruta sea correcta! No debe ejecutar ningún error "Error al resolver: com.facebook.react: reaccionar-native: 0.xx" después de ejecutar la sincronización de Gradle en Android Studio.

Configurando permisos

A continuación, asegúrese de tener el permiso de Internet en su `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Si necesita acceder al `DevSettingsActivity` agregar a su `AndroidManifest.xml`:

```
<activity android:name="com.facebook.react.devsupport.DevSettingsActivity" />
```

Esto solo se usa realmente en el modo dev al volver a cargar JavaScript desde el servidor de desarrollo, por lo que puede quitar esto en compilaciones de versiones si es necesario.

Integración de código

Ahora realmente modificaremos la aplicación nativa de Android para integrar React Native.

El componente React Native

El primer fragmento de código que escribiremos es el código Reaccionar nativo real para la nueva pantalla "Puntuación alta" que se integrará en nuestra aplicación.

1. Crea un `index.js` archivo

Primero, crea un `index.js` archivo vacío en la raíz de tu proyecto React Native.

`index.js` es el punto de partida para las aplicaciones React Native, y siempre es necesario. Puede ser un archivo pequeño que sea `require` otro archivo que sea parte de su componente o aplicación React Native, o puede contener todo el código que se necesita para él. En nuestro caso, nos limitaremos a poner todo en `index.js`.

2. Agregue su código de React Native

En tu `index.js`, crea tu componente. En nuestro ejemplo aquí, agregaremos un `<Text>`componente simple dentro de un estilo `<View>`:

```
'use strict';

import React from 'react';
import {
        AppRegistry,
        StyleSheet,
        Text,
        View
} from 'react-native';

class HelloWorld extends React.Component {
        render() {
                return (
                        <View style={styles.container}>
                                <Text style={styles.hello}>Hello, World</Text>
                        </View>
                )
        }
}
var styles = StyleSheet.create({
        container: {
                flex: 1,
                justifyContent: 'center',
        },
        hello: {
                fontSize: 20,
                textAlign: 'center',
                margin: 10,
```

```
        },
});

AppRegistry.registerComponent('MyReactNativeApp', () => HelloWorld);
```

3. Configure los permisos para la superposición de errores de desarrollo

Si su aplicación apunta a Android `API level 23`o superior, asegúrese de tener el `overlay` permiso habilitado para la compilación de desarrollo. Puedes verificarlo con `Settings.canDrawOverlays(this);`. Esto se requiere en compilaciones de desarrollo porque los errores de desarrollo de reacción nativos deben mostrarse sobre todas las otras ventanas. Debido al nuevo sistema de permisos introducido en el nivel API 23, el usuario debe aprobarlo. Esto se puede lograr agregando el siguiente código al archivo Activity en el método onCreate (). OVERLAY_PERMISSION_REQ_CODE es un campo de la clase que se encargaría de pasar el resultado a la Actividad.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (!Settings.canDrawOverlays(this)) {
                Intent intent = new Intent(Settings.ACTION_MANAGE_OVERLAY_PERMISSION, Uri.parse("package:" +
getPackageName()));
                startActivityForResult(intent, OVERLAY_PERMISSION_REQ_CODE);
        }
}
```

Finalmente, el `onActivityResult()` método (como se muestra en el código a continuación) debe anularse para manejar los casos de Permiso aceptado o Denegado para UX coherente.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == OVERLAY_PERMISSION_REQ_CODE) {
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                        if (!Settings.canDrawOverlays(this)) {
                                // SYSTEM_ALERT_WINDOW permission not granted…
                        }
                }
        }
}
```

La magia: `ReactRootView`

Necesita agregar un código nativo para iniciar el tiempo de ejecución Reaccionar nativo y hacer que muestre algo. Para hacer esto, vamos a crear un `Activity` que crea un `ReactRootView`, inicia una aplicación React dentro de él y lo establece como la vista de contenido principal.

> Si apuntas a la versión de Android <5, utiliza la `AppCompatActivity`clase del `com.android.support:appcompat`paquete en lugar de `Activity`.

```
public class MyReactActivity extends Activity implements DefaultHardwareBackBtnHandler {
        private ReactRootView mReactRootView;
        private ReactInstanceManager mReactInstanceManager;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
```

```
                super.onCreate(savedInstanceState);
                mReactRootView = new ReactRootView(this);
                mReactInstanceManager = ReactInstanceManager.builder() .setApplication(getApplication())
                                        .setBundleAssetName("index.android.bundle")
                                        .setJSMainModulePath("index")
                                        .addPackage(new MainReactPackage())
                                        .setUseDeveloperSupport(BuildConfig.DEBUG)
                                        .setInitialLifecycleState(LifecycleState.RESUMED)
                                        .build();
                mReactRootView.startReactApplication(mReactInstanceManager, "MyReactNativeApp", null);
                setContentView(mReactRootView);
        }

        @Override
        public void invokeDefaultOnBackPressed() {
                super.onBackPressed();
        }
}
```

Si está utilizando un kit de inicio para React Native, reemplace la cadena "HelloWorld" con la que está en su archivo index.js (es el primer argumento para el `AppRegistry.registerComponent()` método).

Si está utilizando Android Studio, use `Alt + Enter` para agregar todas las importaciones faltantes en su clase MyReactActivity. Tenga cuidado de usar su paquete `BuildConfig` y no el del `...facebook...` paquete.

Necesitamos establecer el tema de `MyReactActivity`a `Theme.AppCompat.Light.NoActionBar` porque algunos componentes dependen de este tema.

```
<activity
        android:name=".MyReactActivity"
        android:label="@string/app_name"
        android:theme="@style/Theme.AppCompat.Light.NoActionBar">
</activity>
```

A `ReactInstanceManager`se puede compartir entre múltiples actividades y / o fragmentos. Usted tendrá que hacer su propio `ReactFragment`o `ReactActivity`y tienen un producto único *titular* que mantiene una `ReactInstanceManager`. Cuando necesite `ReactInstanceManager`(por ejemplo, conectar `ReactInstanceManager`el ciclo de vida de esas Actividades o Fragmentos) use el proporcionado por el singleton.

A continuación, debemos pasar algunas llamadas de ciclo de vida de actividad a `ReactInstanceManager`:

```
@Override
protected void onPause() {
        super.onPause();
        if (mReactInstanceManager != null) {
                mReactInstanceManager.onHostPause(this);
        }
}

@Override
protected void onResume() {
        super.onResume();
        if (mReactInstanceManager != null) {
                mReactInstanceManager.onHostResume(this, this);
        }
}

@Override
protected void onDestroy() {
        super.onDestroy();
        if (mReactInstanceManager != null) {
                mReactInstanceManager.onHostDestroy();
        }
```

```
}
```

También necesitamos pasar los eventos de botón a React Native:

```
@Override
public void onBackPressed() {
        if (mReactInstanceManager != null) {
                mReactInstanceManager.onBackPressed();
        } else {
                super.onBackPressed();
        }
}
```

Esto permite a JavaScript controlar lo que sucede cuando el usuario presiona el botón de retroceso de hardware (por ejemplo, para implementar la navegación). Cuando JavaScript no maneja una presión atrás, `invokeDefaultOnBackPressed` se llamará a su método. Por defecto esto simplemente termina tu `Activity`. Finalmente, necesitamos conectar el menú de desarrollo. Por defecto, esto es activado por (rabia) sacudiendo el dispositivo, pero esto no es muy útil en emuladores. Entonces lo hacemos mostrar cuando presiona el botón de menú de hardware (úselo `Ctrl + M` si está usando el emulador de Android Studio):

```
@Override
public boolean onKeyUp(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_MENU && mReactInstanceManager != null){
                mReactInstanceManager.showDevOptionsDialog();
                return true;
        }
        return super.onKeyUp(keyCode, event);
}
```

Ahora su actividad está lista para ejecutar algunos códigos JavaScript.

Pon a prueba tu integración

Ya ha realizado todos los pasos básicos para integrar React Native con su aplicación actual. Ahora comenzaremos el `index.bundle` paquete React Native para compilar el paquete y el servidor que se ejecuta en el servidor local para servirlo.

1. Ejecute el paquete

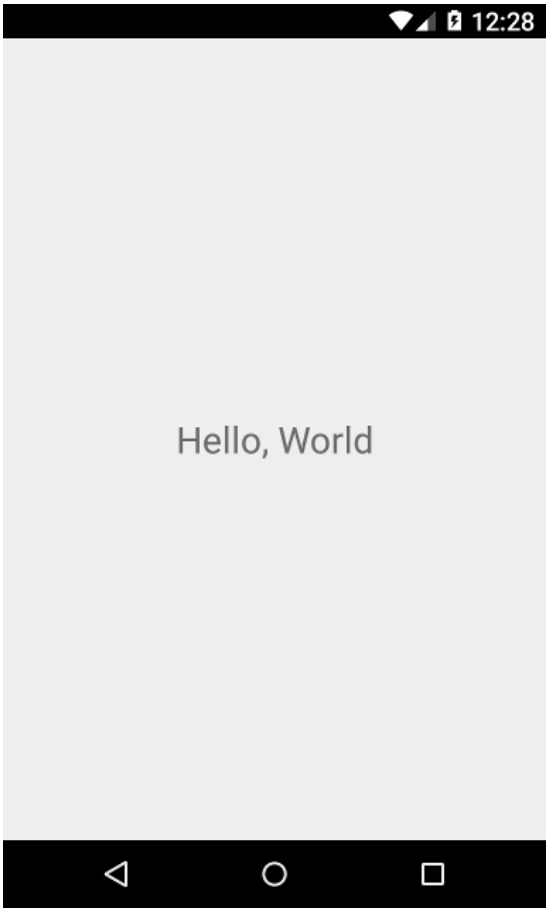
Para ejecutar su aplicación, primero debe iniciar el servidor de desarrollo. Para hacer esto, simplemente ejecute el siguiente comando en el directorio raíz de su proyecto React Native:

```
$ npm start
```

2. Ejecute la aplicación

Ahora compila y ejecuta tu aplicación de Android como siempre.

Una vez que llegue a su actividad de React dentro de la aplicación, debería cargar el código JavaScript del servidor de desarrollo y mostrar:



Crear una compilación de lanzamiento en Android Studio

¡También puedes usar Android Studio para crear tus versiones de lanzamiento! Es tan fácil como crear compilaciones de lanzamiento de su aplicación nativa de Android previamente existente. Solo hay un paso adicional, que tendrás que hacer antes de cada compilación de lanzamiento. Debes ejecutar lo siguiente para crear un paquete React Native, que se incluirá con tu aplicación nativa de Android:

```
$ react-native bundle --platform android --dev false --entry-file index.js --bundle-output android/com/your-company-name/app-package-name/src/main/assets/index.android.bundle --assets-dest android/com/your-company-name/app-package-name/src/main/res/
```

No olvides reemplazar las rutas con las correctas y crear la carpeta de activos si no existe.

Ahora solo crea una compilación de lanzamiento de tu aplicación nativa desde Android Studio como siempre, ¡y deberías estar listo para empezar!

¿Ahora que?

En este punto, puedes continuar desarrollando tu aplicación como siempre. Consulte nuestros documentos de depuración e implementación para obtener más información sobre cómo trabajar con React Native.

Ejecutando en el dispositivo

Proyecto con código nativo requerido

Esta página solo se aplica a proyectos realizados con `react-native init` o creados con la aplicación Create React Native que se han expulsado. Para obtener más información sobre la expulsión, consulte la guía en el repositorio Crear repositorio nativo de la aplicación.

Siempre es una buena idea probar su aplicación en un dispositivo real antes de liberarla a sus usuarios. Este documento lo guiará a través de los pasos necesarios para ejecutar su aplicación React Native en un dispositivo y prepararlo para la producción.

Si usó la aplicación Create React Native para configurar su proyecto, puede obtener una vista previa de su aplicación en un dispositivo escaneando el código QR con la aplicación Expo. Para construir y ejecutar su aplicación en un dispositivo, deberá expulsar e instalar las dependencias del código nativo de la guía de introducción .

- iOS

Ejecutando su aplicación en dispositivos iOS

SO de desarrollo: MacOS

1. Conecte su dispositivo a través de USB

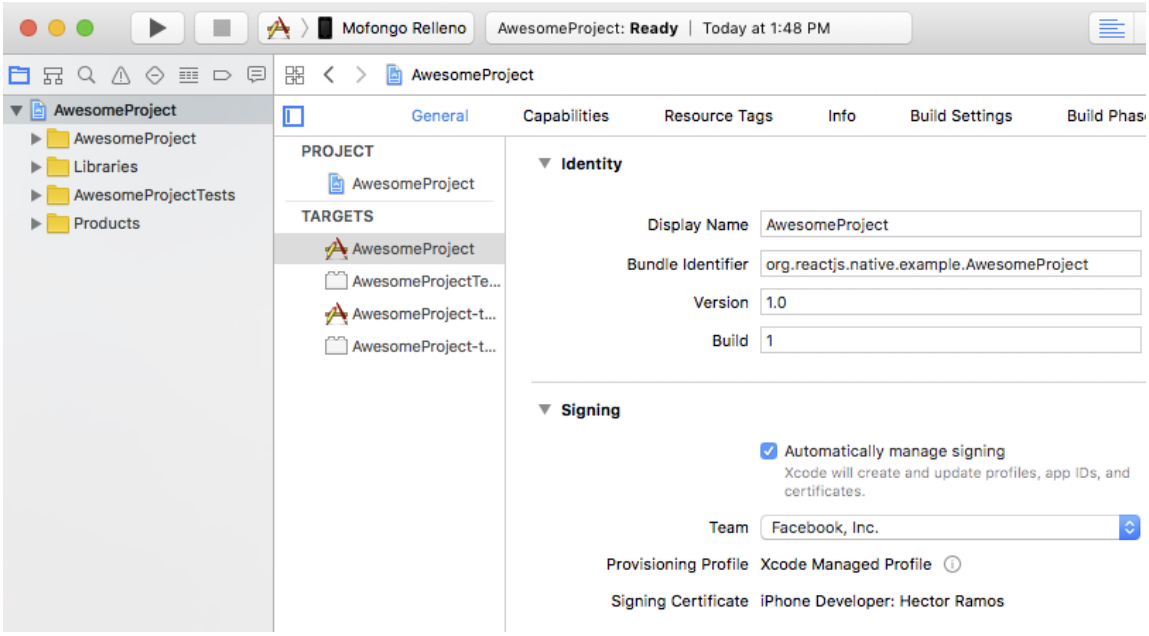
Conecte su dispositivo iOS a su Mac usando un cable USB a Lightning. Navega a la `ios` carpeta en tu proyecto, luego abre el `.xcodeproj` archivo dentro de ella usando Xcode.

Si esta es la primera vez que ejecuta una aplicación en su dispositivo iOS, es posible que deba registrar su dispositivo para el desarrollo. Abra el menú **Producto** en la barra de menú de Xcode, luego vaya a **Destino** . Busque y seleccione su dispositivo de la lista. Xcode luego registrará su dispositivo para el desarrollo.

2. Configure la firma del código

Regístrese para una cuenta de desarrollador de Apple si aún no tiene una.

Seleccione su proyecto en Xcode Project Navigator, luego seleccione su objetivo principal (debe compartir el mismo nombre que su proyecto). Busque la pestaña "General". Vaya a "Firmar" y asegúrese de que su cuenta o equipo de desarrollador de Apple esté seleccionado en el menú desplegable de Equipo.



Repita este paso para el objetivo de Pruebas en su proyecto.

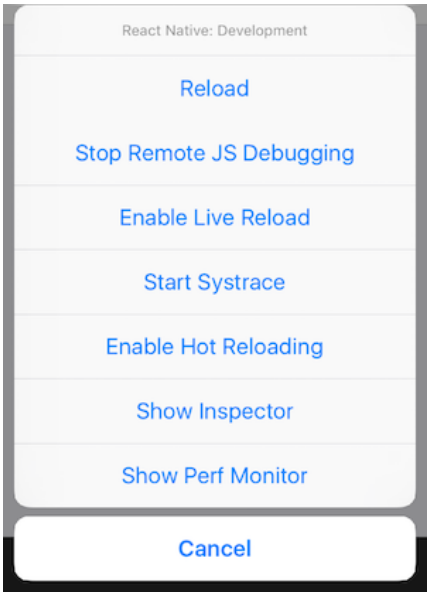
3. Crea y ejecuta tu aplicación

Si todo está configurado correctamente, su dispositivo aparecerá como el objetivo de compilación en la barra de herramientas de Xcode, y también aparecerá en el panel Dispositivos (⇧⌘2). Ahora puede presionar el botón **Generar y ejecutar** (⌘R) o seleccionar **Ejecutar** en el menú **Producto** . Su aplicación se iniciará en su dispositivo en breve.

Si se encuentra con algún problema, eche un vistazo a Apple's Launching App en un dispositivo de documentos.

Conectando al servidor de desarrollo

También puede iterar rápidamente en un dispositivo usando el servidor de desarrollo. Solo debe estar en la misma red Wi-Fi que su computadora. Agite su dispositivo para abrir el menú de Desarrollador , luego habilite Live Reload. Su aplicación se volverá a cargar cada vez que su código JavaScript haya cambiado.



Resolución de problemas

> Si tiene algún problema, asegúrese de que su Mac y su dispositivo estén en la misma red y puedan comunicarse entre sí. Muchas redes inalámbricas abiertas con portales cautivos están configuradas para evitar que los dispositivos lleguen a otros dispositivos en la red. En este caso, puede usar la función de Hotspot personal de su dispositivo.

Al intentar conectar con el servidor de desarrollo, es posible que aparezca una pantalla roja con un error que dice:

> Conexión a http: // localhost: 8081 / debugger-proxy? role = tiempo de espera del cliente agotado. ¿Estás ejecutando proxy de nodo? Si está ejecutando en el dispositivo, verifique si tiene la dirección IP correcta en `RCTWebSocketExecutor.m`.

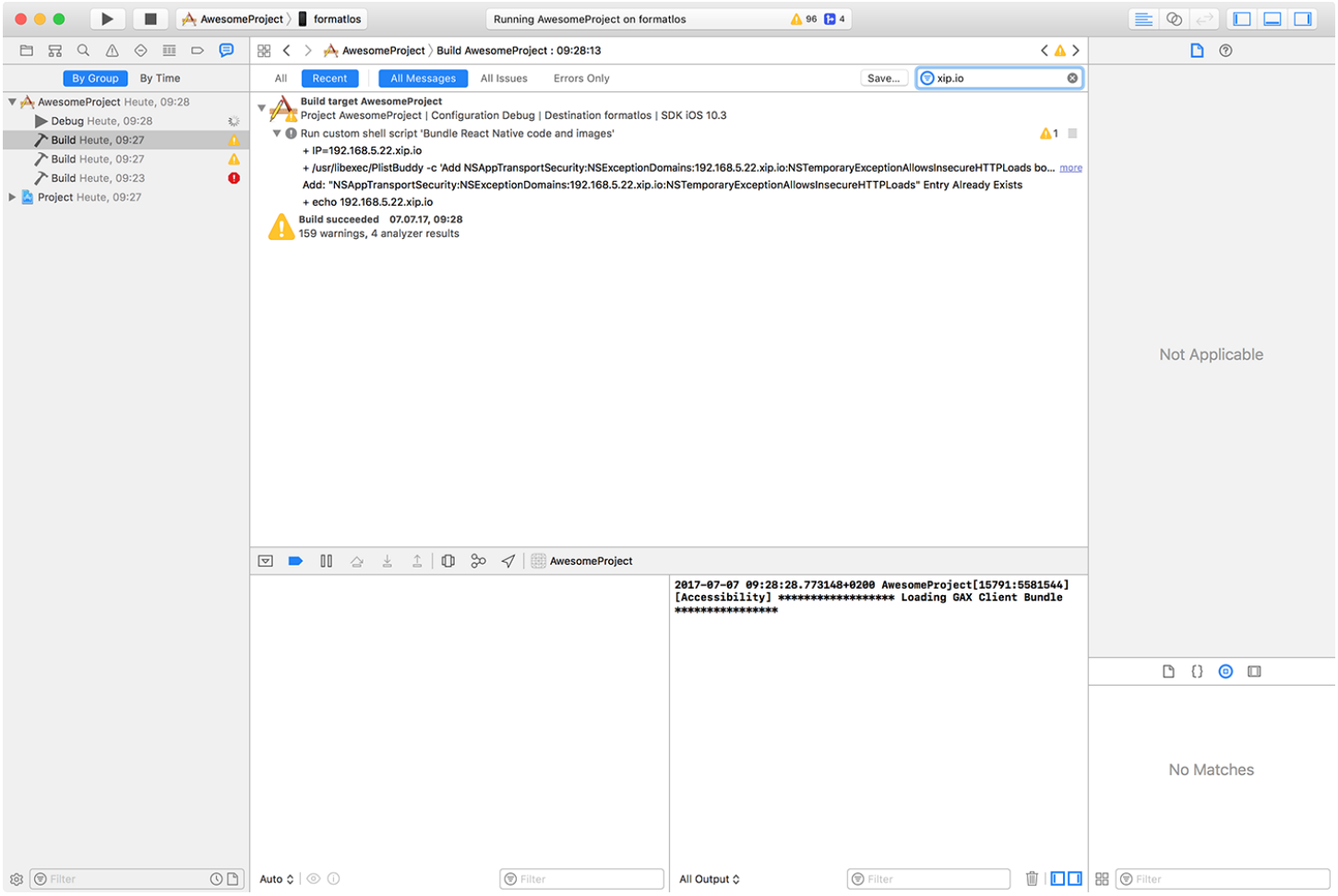
Para resolver este problema, compruebe los siguientes puntos.

1. red Wi-Fi.

Asegúrese de que su computadora portátil y su teléfono estén en la **misma** red Wi-Fi.

2. dirección IP

Asegúrese de que el script de compilación haya detectado correctamente la dirección IP de su máquina (p. Ej., 10.0.1.123).



Abra la pestaña **Navegador de informes** , seleccione la última **compilación** y busque `xip.io`. La dirección IP que se integra en la aplicación debe coincidir con la dirección IP de la máquina más el dominio `.xip.io`(por ejemplo, 10.0.1.123.xip.io)

3. Configuración de red / enrutador

React Native usa el servicio DNS comodín **xip.io** para direccionar su dispositivo. Algunos enrutadores tienen características de seguridad para evitar que los servidores DNS resuelvan algo en el rango de IP local.

Ahora compruebe si puede resolver la dirección xip.io ejecutando `nslookup`.

```
$ nslookup 10.0.1.123.xip.io
```

Si no resuelve su dirección IP local, el servicio **xip.io** está inactivo o es más probable que su enrutador lo impida.

Para seguir usando xip.io detrás de tu rooter:

- configura tu teléfono para usar Google DNS (8.8.8.8)
- desactivar la función de seguridad adecuada en su enrutador
-

Creando su aplicación para producción

Has creado una gran aplicación con React Native, y ahora estás ansioso por lanzarla en la App Store. El proceso es el mismo que el de cualquier otra aplicación iOS nativa, con algunas consideraciones adicionales a tener en cuenta.

1. Habilitar seguridad de transporte de aplicaciones

App Transport Security es una característica de seguridad presentada en iOS 9 que rechaza todas las solicitudes HTTP que no se envían a través de HTTPS. Esto puede provocar el bloqueo del tráfico HTTP, incluido el servidor React Native del desarrollador. ATS está deshabilitado `localhost` por defecto en los proyectos React Native para facilitar el desarrollo.

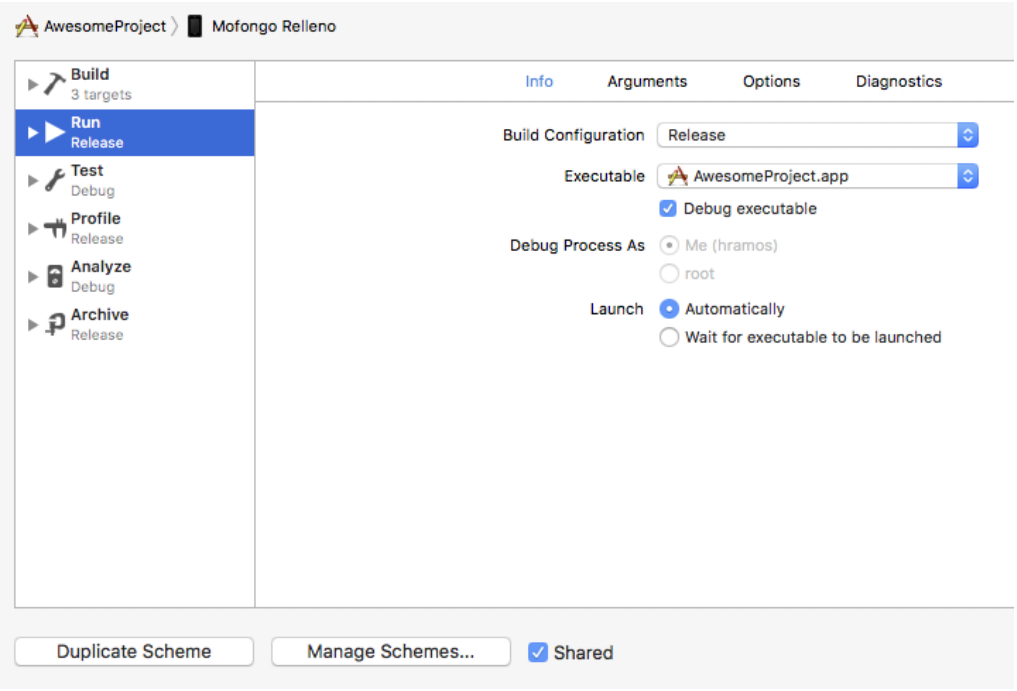
Debe volver a habilitar ATS antes de crear su aplicación para producción eliminando la `localhost` entrada del `NSExceptionDomains` diccionario en su `Info.plist` archivo en la `ios/`carpeta. También puede volver a habilitar ATS desde Xcode abriendo sus propiedades de destino en el panel de información y editando la entrada de configuración de seguridad de transporte de la aplicación.

> Si su aplicación necesita acceder a recursos HTTP en producción, consulte esta publicación para aprender a configurar ATS en su proyecto.

2. Configure el esquema de liberación

Crear una aplicación para distribución en la App Store requiere usar el `Release` esquema en Xcode. Las aplicaciones creadas `Release` desactivarán automáticamente el menú del desarrollador en la aplicación, lo que evitará que los usuarios accedan inadvertidamente al menú en producción. También incluirá el JavaScript localmente, para que pueda poner la aplicación en un dispositivo y probarla mientras no esté conectado a la computadora.

Para configurar su aplicación para que se cree utilizando el `Release` esquema, vaya a **Producto → Esquema → Editar esquema** . Seleccione la pestaña **Ejecutar**en la barra lateral, luego configure el menú desplegable Configuración de construcción en `Release`.



3. Crea la aplicación para el lanzamiento

Ahora puede crear su aplicación para su lanzamiento tocando ⌘Bo seleccionando **Producto** → **Crear** en la barra de menú. Una vez creado para su lanzamiento, podrá distribuir la aplicación a los probadores beta y enviarla a App Store.

> También puede usar el `React Native CLI` para realizar esta operación usando la opción `–configuration` con el valor `Release` (por ejemplo `react-native run-ios --configuration Release`).

Androide

Ejecutando su aplicación en dispositivos Android

SO de desarrollo: MacOS Linux Windows

1. Habilite la depuración sobre USB

La mayoría de los dispositivos Android solo pueden instalar y ejecutar aplicaciones descargadas de Google Play de forma predeterminada. Deberá habilitar la depuración de USB en su dispositivo para instalar su aplicación durante el desarrollo.

Para habilitar la depuración de USB en su dispositivo, primero deberá habilitar el menú "Opciones de desarrollador" yendo a **Configuración** → **Acerca del teléfono** y luego tocando la `Build number` fila en la parte inferior siete veces. A continuación, puede volver a **Configuración** → **Opciones de desarrollador** para habilitar "Depuración de USB".

2. Conecte su dispositivo a través de USB

Vamos a configurar un dispositivo Android para ejecutar nuestros proyectos React Native. Continúe y conecte su dispositivo a través de USB a su máquina de desarrollo.

Ahora compruebe que su dispositivo se está conectando correctamente a ADB, el Android Debug Bridge, ejecutándose `adb devices`.

```
$ adb devices
List of devices attached
emulator-5554 offline # Google emulator
14ed2fcc device # Physical device
```

Ver `device` en la columna de la derecha significa que el dispositivo está conectado. Debe tener **solo un dispositivo conectado** a la vez.

3. Ejecute su aplicación

Escriba lo siguiente en el símbolo del sistema para instalar e iniciar su aplicación en el dispositivo:

```
$ react-native run-android
```

> Si obtiene un error de "la configuración del puente no está disponible", consulte Uso de adb reverse .
>
> Insinuación
>
> También puede usar el `React Native CLI`para generar y ejecutar una `Release`construcción (por ejemplo `react-native run-android –variant=release`).

Conectando al servidor de desarrollo

También puede iterar rápidamente en un dispositivo conectándose al servidor de desarrollo que se ejecuta en su máquina de desarrollo. Hay varias maneras de lograr esto, dependiendo de si tiene acceso a un cable USB o a una red Wi-Fi.

Método 1: usar adb reverse (recomendado)

Puede utilizar este método si su dispositivo ejecuta Android 5.0 (Lollipop) o posterior, tiene habilitada la depuración de USB y está conectado a través de USB a su máquina de desarrollo.

Ejecute lo siguiente en un símbolo del sistema:

```
$ adb reverse tcp:8081 tcp:8081
```

Ahora puede habilitar la recarga en vivo desde el menú Desarrollador . Su aplicación se volverá a cargar cada vez que su código JavaScript haya cambiado.

Método 2: conectarse a través de Wi-Fi

También puede conectarse al servidor de desarrollo a través de Wi-Fi. Primero deberá instalar la aplicación en su dispositivo con un cable USB, pero una vez que lo haya hecho, puede depurar de manera inalámbrica siguiendo estas instrucciones. Necesitará la dirección IP actual de su máquina de desarrollo antes de continuar.

Abra el símbolo del sistema y escriba `ipconfig` para encontrar la dirección IP de su máquina (más información).

1. Asegúrese de que su computadora portátil y su teléfono estén en la **misma** red Wi-Fi.
2. Abra su aplicación React Native en su dispositivo.
3. Verás una pantalla roja con un error . Esto esta bien. Los siguientes pasos lo arreglarán.
4. Abra el menú del desarrollador en la aplicación .
5. Vaya a **Dev Settings → Debug server host para el dispositivo** .
6. Escriba la dirección IP de su máquina y el puerto del servidor de desarrollo local (por ejemplo, 10.0.1.1:8081).
7. Regrese al **menú Desarrollador** y seleccione **Recargar JS** .

Ahora puede habilitar la recarga en vivo desde el menú Desarrollador . Su aplicación se volverá a cargar cada vez que su código JavaScript haya cambiado.

Creando su aplicación para producción

Has creado una excelente aplicación con React Native, y ahora estás ansioso por lanzarla en Play Store. El proceso es el mismo que el de cualquier otra aplicación nativa de Android, con algunas consideraciones adicionales a tener en cuenta. Siga la guía para generar un APK firmado para obtener más información.

Actualizando a las nuevas versiones de React Native

La actualización a nuevas versiones de React Native le dará acceso a más API, vistas, herramientas de desarrollo y otras ventajas. Actualizar requiere una pequeña cantidad de esfuerzo, pero tratamos de hacerlo más fácil para usted. Las instrucciones son un poco diferentes dependiendo de si usó `create-react-native-app`o `react-native init` para crear su proyecto.

Crear proyectos de la aplicación React Native

Actualización de su proyecto Crear Reaccionar aplicación nativa para una nueva versión de React nativo requiere la actualización de los `react-native`, `react` y `expo` versiones de los paquetes en el `package.json` archivo. Consulte este documento para averiguar qué versiones son compatibles. También necesitará establecer el correcto `sdkVersion` en su `app.json` archivo.

Consulte la guía del usuario de CRNA para obtener información actualizada sobre la actualización de su proyecto.

Proyectos construidos con código nativo

Proyectos con código nativo solamente

Esta sección solo se aplica a proyectos hechos con `react-native init` o creados con la aplicación nativa Create React que se han expulsado desde entonces. Para obtener más información sobre la expulsión, consulte la guía en el repositorio Crear repositorio nativo de la aplicación.

Debido a que los proyectos de React Native construidos con código nativo se componen esencialmente de un proyecto de Android, un proyecto de iOS y un proyecto de JavaScript, la actualización puede ser bastante complicada. Esto es lo que debe hacer para actualizar desde una versión anterior de React Native.

Actualización basada en Git

El módulo `react-native-git-upgrade` proporciona una operación de un solo paso para actualizar los archivos fuente con un mínimo de conflictos. Debajo del capó, consiste en 2 fases:

- En primer lugar, calcula un parche Git entre archivos de plantilla antiguos y nuevos,
- Luego, el parche se aplica a las fuentes del usuario.

IMPORTANTE: no tiene que instalar la nueva versión del `react-native` paquete, se instalará automáticamente.

1. Instalar Git

Si bien su proyecto no tiene que ser manejado por el sistema de control de versiones de Git (puede usar Mercurial, SVN o nada), aún necesitará instalar Git en su sistema para poder usarlo `react-native-git-upgrade`. Git también deberá estar disponible en `PATH`.

2. Instalar el `react-native-git-upgrade`módulo

El `react-native-git-upgrade` módulo proporciona una CLI y debe instalarse globalmente:

```
$ npm install -g react-native-git-upgrade
```

3. Ejecute el comando

Ejecute el siguiente comando para iniciar el proceso de actualización a la última versión:

```
$ react-native-git-upgrade
```

Puede especificar una versión React Native pasando un argumento: `react-native-git-upgrade X.Y`

Las plantillas se actualizan de forma optimizada. Aún puede encontrar conflictos, pero solo en los casos en que falle la combinación de 3 vías de Git, según la versión y la forma en que modificó sus fuentes.

4. Resuelva los conflictos

Los archivos en conflicto incluyen delimitadores que dejan muy claro de dónde provienen los cambios. Por ejemplo:

```
13B07F951A680F5B00A75B9A /* Release */ = {
        isa = XCBuildConfiguration;
        buildSettings = {
                ASSETCATALOG_COMPILER_APPICON_NAME = AppIcon;
<<<<<<< ours
                CODE_SIGN_IDENTITY = "iPhone Developer";
                FRAMEWORK_SEARCH_PATHS = (
                        "$(inherited)",
                        "$(PROJECT_DIR)/HockeySDK.embeddedframework",
                        "$(PROJECT_DIR)/HockeySDK-iOS/HockeySDK.embeddedframework",
                );
=======
        CURRENT_PROJECT_VERSION = 1;
>>>>>>> theirs
        HEADER_SEARCH_PATHS = (
                "$(inherited)",
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include,
                "$(SRCROOT)/../node_modules/react-native/React/**",
                "$(SRCROOT)/../node_modules/react-native-code-push/ios/CodePush/**",
        );
```

Puedes pensar en "nuestro" como "tu equipo" y "de ellos" como "el equipo de desarrollo de React Native".

Alternativa

Úselo solo en caso de que lo anterior no haya funcionado.

1. Actualiza la `react-native` dependencia

Tenga en cuenta la última versión del `react-native` paquete npm desde aquí (o use `npm info react-native` para verificar).

Ahora instala esa versión de `react-native` en tu proyecto con `npm install --save`:

```
$ npm install --save react-native@X.Y
# where X.Y is the semantic version you are upgrading to
npm WARN peerDependencies The peer dependency react@~R included from react-native…
```

Si vio una advertencia sobre peerDependency, también actualice `react`ejecutando:

```
$ npm install --save react@R
# where R is the new version of react from the peerDependency warning you saw
```

2. Actualiza tus plantillas de proyecto

El nuevo paquete npm puede contener actualizaciones de los archivos que normalmente se generan cuando se ejecuta `react-native init`, como los subproyectos iOS y Android.

Puede consultar rn-diff para ver si hubo cambios en los archivos de la plantilla del proyecto. En caso de que no haya ninguno, simplemente reconstruya el proyecto y continúe desarrollando. En caso de cambios menores, puede actualizar su proyecto manualmente y reconstruir.

Si hubo cambios importantes, ejecute esto en una terminal para obtener estos:

```
$ react-native upgrade
```

Esto verificará sus archivos con la plantilla más reciente y realizará lo siguiente:

- Si hay un nuevo archivo en la plantilla, simplemente se crea.
- Si un archivo en la plantilla es idéntico a su archivo, se salta.

- Si un archivo es diferente en su proyecto que la plantilla, se le preguntará; tiene opciones para guardar su archivo o sobrescribirlo con la versión de la plantilla.
-

Actualizaciones manuales

Algunas actualizaciones requieren pasos manuales, por ejemplo, de 0,13 a 0,14 o de 0,28 a 0,29. Asegúrese de revisar las notas de la versión cuando actualice para que pueda identificar cualquier cambio manual que requiera su proyecto en particular.

Resolución de problemas

Estos son algunos de los problemas comunes con los que se puede encontrar al configurar React Native. Si encuentra algo que no se encuentra aquí, intente buscar el problema en GitHub .

Puerto ya en uso

El empaquetador React Native se ejecuta en el puerto 8081. Si otro proceso ya está utilizando ese puerto, puede finalizar ese proceso o cambiar el puerto que usa el empaquetador.

Terminar un proceso en el puerto 8081

Ejecute el siguiente comando en una Mac para encontrar la identificación para el proceso que está escuchando en el puerto 8081:

```
$ sudo lsof -i :8081
```

A continuación, ejecute lo siguiente para finalizar el proceso:

```
$ kill -9 <PID>
```

En Windows puede encontrar el proceso usando el puerto 8081 usando el Monitor de recursos y detenerlo usando el Administrador de tareas.

Usando un puerto que no sea 8081

Puede configurar el empaquetador para usar un puerto que no sea 8081 utilizando el `port` parámetro:

```
$ react-native start -port=8088
```

También deberá actualizar sus aplicaciones para cargar el paquete de JavaScript desde el nuevo puerto. Si se ejecuta en un dispositivo de Xcode, puede hacerlo actualizando las apariciones de `8081`su puerto elegido en el `node_modules/react-native/React/React.xcodeproj/project.pbxproj`archivo.

Error de bloqueo de NPM

Si encuentra un error tal como `npm WARN locking Error: EACCES`al usar la CLI de React Native, intente ejecutar lo siguiente:

```
sudo chown -R $USER ~/.npm
sudo chown -R $USER /usr/local/lib/node_modules
```

Bibliotecas faltantes para Reaccionar

Si ha añadido Reaccionar nativo manualmente a su proyecto, asegúrese de que ha incluido todas las dependencias pertinentes que está utilizando, como `RCTText.xcodeproj`, `RCTImage.xcodeproj`. A continuación, los binarios creados por estas dependencias deben estar vinculados a su aplicación binaria. Use la `Linked Frameworks and Binaries` sección en la configuración del proyecto Xcode. Pasos más detallados están aquí: Linking Libraries .

Si está utilizando CocoaPods, verifique que ha agregado React junto con las subespecies a `Podfile`. Por ejemplo, si estuviera usando los `<Text />`, `<Image />` y `fetch()` las API, lo que tendría que añadir estos en su `Podfile`:

```
pod 'React', :path => '../node_modules/react-native', :subspecs => [
        'RCTText',
        'RCTImage',
        'RCTNetwork',
        'RCTWebSocket',
]
```

Luego, asegúrese de ejecutar `pod install` y de que `Pods/` se haya creado un directorio en su proyecto con React instalado. CocoaPods le indicará que use el `.xcworkspace` archivo generado de ahora en adelante para poder usar estas dependencias instaladas.

Lista de argumentos demasiado larga: la expansión de encabezado recursivo falló

En configuración de generación del proyecto, `User Search Header Paths` y `Header Search Paths` son dos configuraciones que especifican donde Xcode debe buscar `#import` los archivos de cabecera especificados en el código. Para los Pods, CocoaPods usa una matriz predeterminada de carpetas específicas para mirar. Verifique que esta configuración particular no se sobrescriba y que ninguna de las carpetas configuradas sea demasiado grande. Si una de las carpetas es una carpeta grande, Xcode intentará buscar de forma recursiva todo el directorio y arrojar el error anterior en algún momento.

Para revertir el `User Search Header Paths` y `Header Search Paths` construir los ajustes a sus valores por defecto establecidos por CocoaPods - seleccione la entrada en el panel de configuración de generación y pulse borrar. Eliminará la anulación personalizada y volverá a los valores predeterminados de CocoaPod.

No hay transportes disponibles

React Native implementa un polyfill para WebSockets. Estos polyfills se inicializan como parte del módulo react-native que incluye en su aplicación `import React from 'react'`. Si carga otro módulo que requiere WebSockets, como Firebase , asegúrese de cargar / requerirlo después de reaccionar-native:

```
import React from 'react';
import Firebase from 'firebase';
```

Shell Command no responde excepción

Si encuentra una excepción ShellCommandUnresponsiveException como, por ejemplo:

```
Execution failed for task ':app:installDebug'.
com.android.builder.testing.api.DeviceException:
com.android.ddmlib.ShellCommandUnresponsiveException
```

Trate La disminución en su versión 1.2.3 a Gradle en `android/build.gradle`.

react-native init cuelga

Si se encuentra con problemas donde la ejecución se `react-native init` cuelga en su sistema, intente ejecutarlo de nuevo en modo detallado y consulte el # 2797 para causas comunes:

```
react-native init --verbose
```

No se puede iniciar el gestor de paquetes react-native (en Linux)

Caso 1: "código" de error: "ENOSPC", "errno": "ENOSPC"

El problema causado por la cantidad de directorios inotify (usados por vigilantes en Linux) puede monitorear. Para resolverlo, simplemente ejecuta este comando en tu ventana de terminal

```
echo fs.inotify.max_user_watches=582222 | sudo tee -a /etc/sysctl.conf && sudo sysctl -p
```