React Native

React Native

What is React Native

---

1. React Native is a JavaScript framework for writing real, natively rendering mobile applications for iOS and Android.
2. React Native applications are written using a mixture of JavaScript and XML markup, known as JSX.
3. React Native "bridge" invokes the native rendering APIs in Objective-C (for iOS) or Java (for Android). Thus, your application will render using real mobile UI components, *not* webviews, and will look and feel like any other mobile application.
4. React Native currently supports both iOS and Android

Why Should I Learn ES6?

React uses ES6, and you should be familiar with some of the new features like:

1. Classes
2. Arrow Functions
3. Variables (let, const, var)
4. Array Methods like .map()
5. Destructuring
6. Modules
7. Ternary Operator
8. Spread Operator

Advantage of React Native

1. The fact that React Native actually renders using its host platform's standard rendering APIs enables it to stand out from most existing methods of cross-platform application development, like Cordova or Ionic. Existing methods of writing mobile applications using combinations of JavaScript, HTML, and CSS typically render using webviews. While this approach can work, it also comes with drawbacks, especially around performance. Additionally, they do not usually have access to the host platform's set of native UI elements. When these frameworks do try to mimic native UI elements, the results usually "feel" just a little off.
2. In contrast, React Native actually translates your markup to real, native UI elements, leveraging existing means of rendering views on whatever platform you are working with.
3. React works separately from the main UI thread, so your application can maintain high performance without sacrificing capability.
4. The update cycle in React Native is the same as in React: when props or state change, React Native re-renders the views.
5. React Native is "just" JavaScript, you don't need to rebuild your application in order to see your changes reflected.
6. React Native lets you take advantage of intelligent debugging tools and error reporting.
7. Code reuse and knowledge

React VS React Native

REACT JS	REACT NATIVE
It is used for developing web applications.	It is used for developing mobile applications.
It uses React-router for navigating web pages.	It has a built-in navigator library for navigating mobile applications.
It uses HTML tags.	It does not use HTML tags.
It provides high security.	It provides low security in comparison to ReactJS.
In this, the virtual DOM renders the browser code.	In this, Native uses its API to render code for mobile applications.

List the essential components of React Native.

These are the core components of React Native:

- **View:** It is the basic built-in component used to build UI of Mobile apps. The view is similar to the div in HTML. It is a content area where you can display your content.
- **States:** It is used to control the components. The variable data can be stored in the state. It is mutable means a state can change the value at any time.
- **Props:** Props are used to pass data to the different components. It is immutable means props cannot change the value. It provides a connection between the container component and a presentation component.
- **Style:** It is an essential component in the web or mobile, which makes the application attractive. React Native does not require any special language or syntax for styling. It can style the application using the JavaScript object.
- **Text:** This component displays text in the app. It uses the basic component textInput to take text input from the user.
- **ScrollView:** It is a scrolling container used to host multiple views. It can be used to render the large list or content in view with a scroll bar.

How many threads run in React Native?

The React Native app contains the following thread:

- **React Native UI Thread (Main Thread):** This thread is used for the layout of the mobile app.
- **React Native JavaScript Thread:** It is a thread where our business logic will run. It means JS thread is a place where our JavaScript code is executed.
- **React Native Modules Thread:** Sometimes, our app needs access to platform API, which happens as a part of native module thread.

- **React Native Render Thread:** This thread is used to generate actual OpenGL commands used to draw the app UI.

What are states in React Native?

It is used to control the components. The variable data can be stored in the state. It is mutable means a state can change the value at any time.

What are props in React Native?

The properties of React Native components are pronounced as props. They are used to pass data to the different components. In React Native, several components are customized at the time of creation with different parameters, and these parameters are known as props. It is immutable means props cannot change the value. It provides a connection between the container component and a presentation componen

What are Refs in React Native?

React refs are useful features that allow you to access DOM elements or component's instance directly within React. It comes handy in situations where you want to change the child of a component without using props or re-rendering the whole component.

What are the differences between Class and Functional Component?

The essential differences between the class component and functional component are:

Syntax: The declaration of both components is different. A functional component takes props, and returns React element, whereas the class components require to extend from React.

```
//Functional Component
function WelcomeMessage(props) {
  return <h1>Welcome to the , {props.name}</h1>;
}

//Class Component
class MyComponent extends React.Component {
  render() {
    return (
```

```
      <div>This is main component.</div>
    );
  }
}
```

State: The class component has a state while the functional component is stateless.

Lifecycle: The class component has a lifecycle, while the functional component does not have a lifecycle.

When would you prefer a class component over functional components?

We prefer class component when the component has a state and lifecycle; otherwise, the functional component should be used.

Functional Components vs Class Components:

Functional Components	Class Components
A functional component is just a plain JavaScript pure function that accepts props as an argument and returns a React element(JSX).	A class component requires you to extend from React. Component and create a render function which returns a React element.
There is no render method used in functional components.	It must have the render() method returning JSX (which is syntactically similar to HTML)
Functional component run from top to bottom and once the function is returned it cant be kept alive.	Class component is instantiated and different life cycle method is kept alive and being run and invoked depending on phase of class component.
Also known as Stateless components as they simply accept data and display them in some form, that they are mainly responsible for rendering UI.	Also known as Stateful components because they implement logic and state.
React lifecycle methods (for example, componentDidMount) cannot be used in functional components.	React lifecycle methods can be used inside class components (for example, componentDidMount).

Hooks can be easily used in functional components to make them Stateful.
example: const [name,SetName]= React.useState(‘ ‘)
	It requires different syntax inside a class component to implement hooks.
	example: constructor(props) {
	super(props);
	this.state = {name: ‘ ‘}
	}

Constructors are not used.	Constructor are used as it needs to store state.

React Hooks

Hooks were added to React in version 16.8.

Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

Although Hooks generally replace class components, there are no plans to remove classes from React.

If you find yourself keeping track of multiple pieces of state that rely on complex logic, useReducer may be useful.

useState

Syntax

The useSate Hook

```
const [color, setColor] = useState("red");
```

The reducer function contains your custom state logic and the initialStatecan be a simple value but generally will contain an object.

eg.

```
import React, { useState } from "react";

function FavoriteColor() {
  const [color, setColor] = useState("red");

  return (
    <>
      <h1>My favorite color is {color}!</h1>
      <button
```

```
        type="button"
        onClick={() => setColor("blue")}
      >Blue</button>
      <button
        type="button"
        onClick={() => setColor("red")}
      >Red</button>
      <button
        type="button"
        onClick={() => setColor("pink")}
      >Pink</button>
      <button
        type="button"
        onClick={() => setColor("green")}
      >Green</button>
    </>
  );

}
```

Hook Rules

There are 3 rules for hooks:

- Hooks can only be called inside React function components.
- Hooks can only be called at the top level of a component.
- Hooks cannot be conditional

Note: Hooks will not work in React class components.

useEffect

The `useEffect` Hook allows you to perform side effects in your components.

Some examples of side effects are: fetching data, directly updating the DOM, and timers.

`useEffect` accepts two arguments. The second argument is optional.

```
useEffect(<function>, <dependency>)
```

Example

```
import { useState, useEffect } from "react";
import ReactDOM from "react-dom/client";

function Timer() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  });

  return <h1>I've rendered {count} times!</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Timer />);
```

`useEffect` runs on every render. That means that when the count changes, a render happens, which then triggers another effect.

This is not what we want. There are several ways to control when side effects run.

We should always include the second parameter which accepts an array. We can optionally pass dependencies to `useEffect` in this array.

There are 3 controls for side effects

1. No dependency passed:

```
useEffect(() => {

  //Runs on every render

});
```

2. An empty array:

```
useEffect(() => {

  //Runs only on the first render

}, []);
```

3. Props or state values:

```
useEffect(() => {

  //Runs on the first render

  //And any time any dependency value changes

}, [prop, state]);
```

Effect Cleanup

Some effects require cleanup to reduce memory leaks.

Timeouts, subscriptions, event listeners, api calls and other effects that are no longer needed should be disposed.

```
useEffect(() => {
    let timer = setTimeout(() => {
    setCount((count) => count + 1);
  }, 1000);
```

```
  return () => clearTimeout(timer)

  }, []);
```

useContext

useRef

useReducer

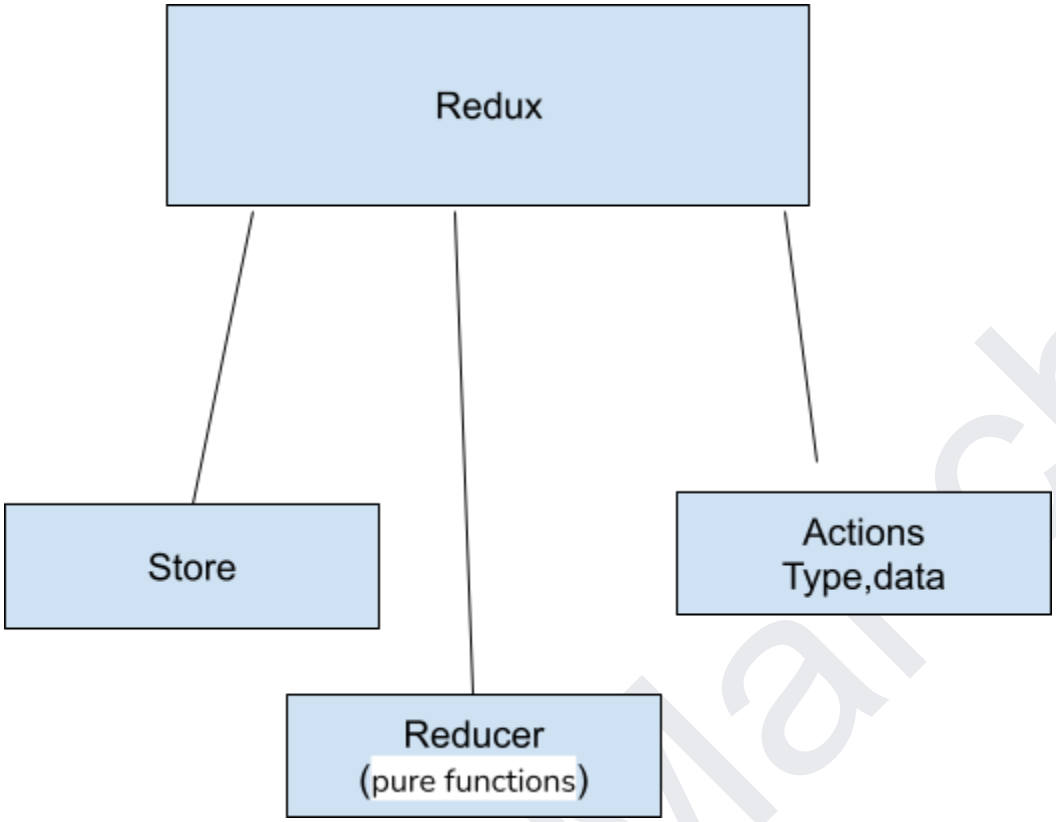
useCallback

useMemo

Custom Hooks

Redux

Redux Key Component pillars



Step To Configure Redux

Step 1: Redux
Step 2: React Redux
Step 3:
- i) Action
- ii) Reducer
- iii) Store

Step 4: Configuration
- i) index.js

```
<Provider store={store}>
      <App />
  </Provider>
AppRegistry.registerComponent(appName, () => ReduxStore);
```

Step 5 :
i) Map props

```
const mapStateToProps = (state) => {
   return {
       //   cards: state.addToCartReducer.cart,
       myCards: state.addToCartReducer.addToMyCart
   }
}
```

ii) Dispatch event

```
const mapDispatchToProps = dispatch => ({
   //addCard: (cart) => dispatch(addToCard(cart)),
   addToMyCart: (cart) => dispatch(addToMyCart(cart))
})
```

iii) Accessing using connect function

```
connect(mapStateToProps, mapDispatchToProps)(PDP)
```

What is Reducer?

A reducer is a function that is able to process our message, our Action. A reducer takes the existing state and applies the message on it. The end result is a new state.

All versions using in my react development

Xcode version 12.5.1
React version 17.0.2
React native version 0.67.3
Node version v16.9.1