Fundamentals of React Native

SCS 3212 - Mobile Applications Development

Pasindu Marasinghe
ppm@ucsc.cmb.ac.lk

Lecture Overview

- JSX
- Components
- State and Props
- Component Life Cycle

JSX

Introduction to JSX

```
const element = <h1>Hello, world!</h1>;
```

- Not HTML, Not a String, Not pure JavaScript.
- A syntax extension to JavaScript, used to describe UI in react/react native.
- JSX is a statically-typed, object-oriented programming language designed to run on modern web browsers.
- XML/HTML-like syntax used by React that extends ECMAScript, so that XML/HTML-like text can coexist with JavaScript/React code.
- Unlike the past, instead of putting JavaScript into HTML, JSX allows us to put HTML into JavaScript.

Characteristics of JSX

- **Faster**
  - JSX performs optimization while compiling the source code to JavaScript.
  - The generated code runs faster than an equivalent code written directly in JavaScript
- **Safer**
  - In contrast to JavaScript, JSX is statically-typed and mostly type-safe.
  - Many errors will be caught during the compilation process.
  - Offers debugging features at the compiler level as well.
- **Easier**
  - JSX offers a solid class system much like Java, freeing the developers from working with the too-primitive prototype-based inheritance system provided by JavaScript.
  - Expressions and statements, however, are mostly equal to JavaScript, so it is easy for JavaScript programmers to start using JSX.

Why JSX?

- Instead of artificially separating technologies by putting markup and logic in separate files, React separates concerns with loosely coupled units called “components” that contain both.
- React embraces the fact that rendering logic is inherently coupled with other UI logic
    - how events are handled
    - how the state changes over time
    - how the data is prepared for display

Components

Components

- A component is a basic element in react-native.
- Using components, large applications can divide into smaller elements.
- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- Components make development fast and code maintenance easier.
- There are two main types of components,
  - Class Components
  - Functional Components

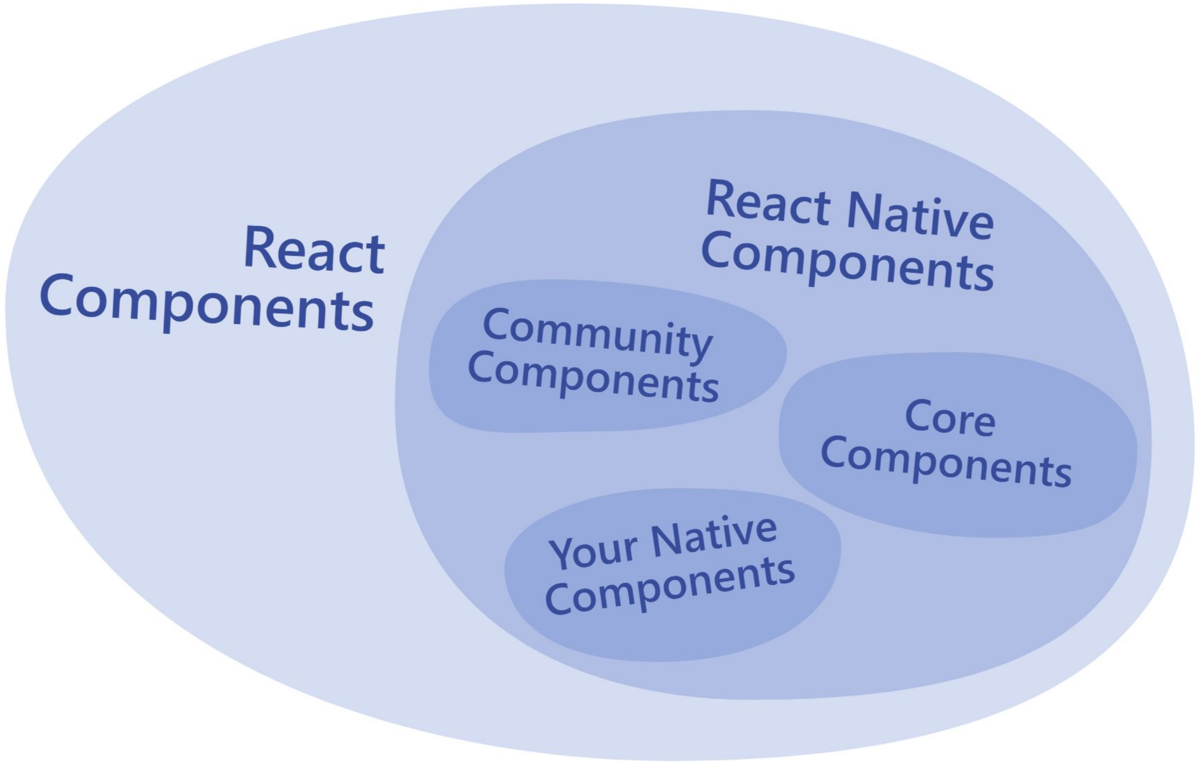
Class Components

- Class components are JavaScript ES2015 classes that extend a base class from React called Component.
- Class components are used as container components to handle state management and wrap child components.
- Has the access to react lifecycle methods
    - i.e render
- Has the access to state/props functionality from the parent
- Class can maintain its own state

Functional Components

- Functional/stateless components are simpler.
- They don’t manage their own state or have access to the lifecycle methods.
- They are literally plain old JavaScript functions, and are sometimes called stateless components.
- Generally used for display purposes –
    - These components call functions from parent components to handle user interactions or state updates.

React Components

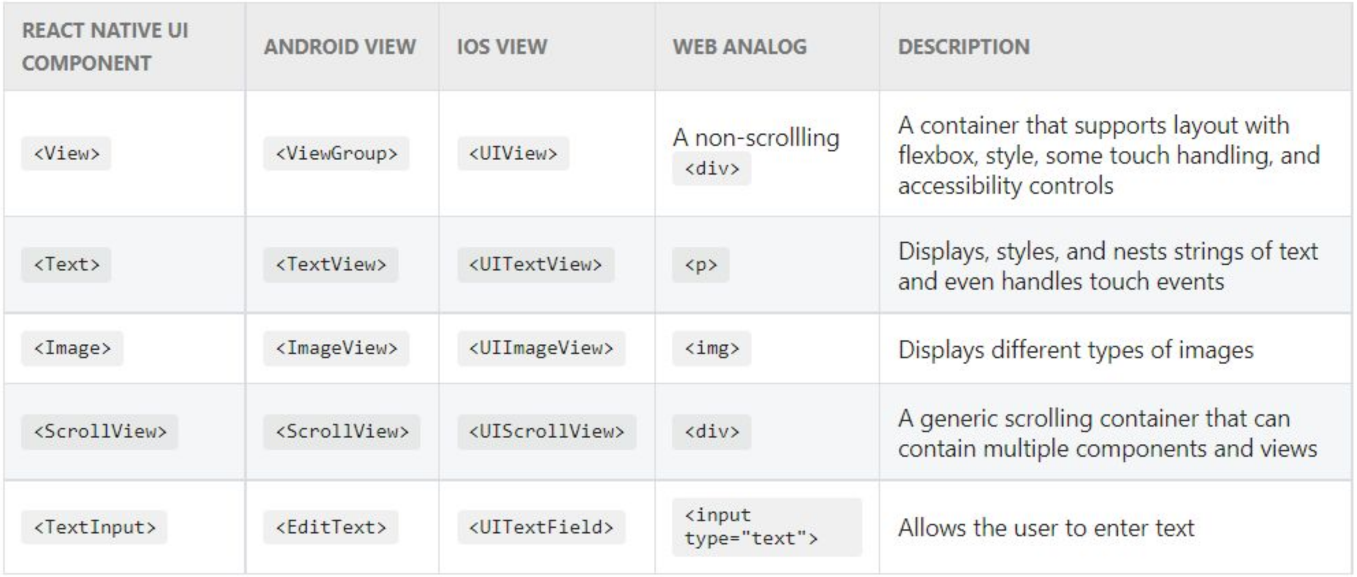


Native Components

- In native mobile application, native components can be implemented in Kotlin or Java for Android, Swift or Objective-C for iOS development.
- With React Native, you can invoke these views with JavaScript using React components.
- At runtime, React Native creates the corresponding Android and iOS views for those components.
- Because React Native components are backed by the same views as Android and iOS, React Native apps look, feel, and perform like any other apps.
- We call these platform-backed components Native Components.
- React Native lets you to build your own Native Components for Android and iOS to suit your app’s unique needs.

React Native Core Components

- React Native includes a set of essential, ready-to-use Native Components.
    - https://reactnative.dev/docs/components-and-apis



REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrollling `<div>`	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Displays, styles, and nests strings of text and even handles touch events
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Displays different types of images
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	A generic scrolling container that can contain multiple components and views
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Allows the user to enter text

State and Props

State

- The state can change/update.
- It means a state can change the value at any time.
- The variable data are stored in state.
- Initialize the state in a constructor and change the value by calling the function ‘setState’ when you want.
- State is handled inside the component.

Props

- It is read only.
- It can be used to pass the data to the different component.
- Props are handled outside of the component.
- Does not need ‘setting’ methods.

Component Life Cycle

React Native Component LifeCycle

A component's lifecycle can be divided into 4 parts:

- **Mounting**
  - an instance of a component is being created
- **Updating**
  - when the React component is born in the browser and grows by receiving new updates.
- **Unmounting**
  - the component is not needed and gets unmounted.
- **Error handling**
  - called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component.

Mounting - constructor()

- It’s called before the component is mounted.
- Mainly use for,
    - Initializing local state by assigning an object to this.state.
    - Binding event handler methods to an instance.

Mounting - getDerivedStateFromProps()

- It's invoked right before calling the render method, both on the initial mount and on subsequent updates.
- It should return an object to update the state, or null to update nothing.
- This method doesn't have access to the component instance.
- This method is fired on every render, regardless of the cause.

Mounting - render()

- Only required method in a class component
- When called, it should examine this.props and this.state and return one of the following types:
    - React elements
    - Arrays and fragments
    - Portals
    - String and numbers
    - Booleans or null
- It should be pure function, meaning that it does not modify component state, it returns the same result each time it's invoked, and it does not directly interact with the browser.

Mounting - componentDidMount()

- Invoked immediately after a component is mounted.
- Initialization that requires DOM nodes should go here.
- If you need to load data from a remote endpoint, this is a good place to instantiate the network request.
- This method is a good place to set up any subscriptions. If you do that, don’t forget to unsubscribe in componentWillUnmount().

Updating - shouldComponentUpdate()

- This is used to let React know if a component’s output is not affected by the current change in state or props.
- The default behavior is to re-render on every state change, and in the vast majority of cases you should rely on the default behavior.
- This method only exists as a performance optimization.

Updating - getSnapshotBeforeUpdate()

- It is invoked right before the most recently rendered output is committed
- Any value returned by this lifecycle will be passed as a parameter to componentDidUpdate().

Updating - componentDidUpdate()

- It is invoked immediately after updating occurs.
- This method is not called for the initial render.
- This can be used as an opportunity to operate on the DOM when the component has been updated.
- This is also a good place to do network requests as long as you compare the current props to previous props (Does not need to perform if the props are not changed).

Unmounting - componentWillUnmount()

- It is invoked immediately before a component is unmounted and destroyed.
- Perform any necessary cleanup in this method, such as,
    - Invalidating timers
    - Canceling network requests
    - Cleaning up any subscriptions that were created in componentDidMount().

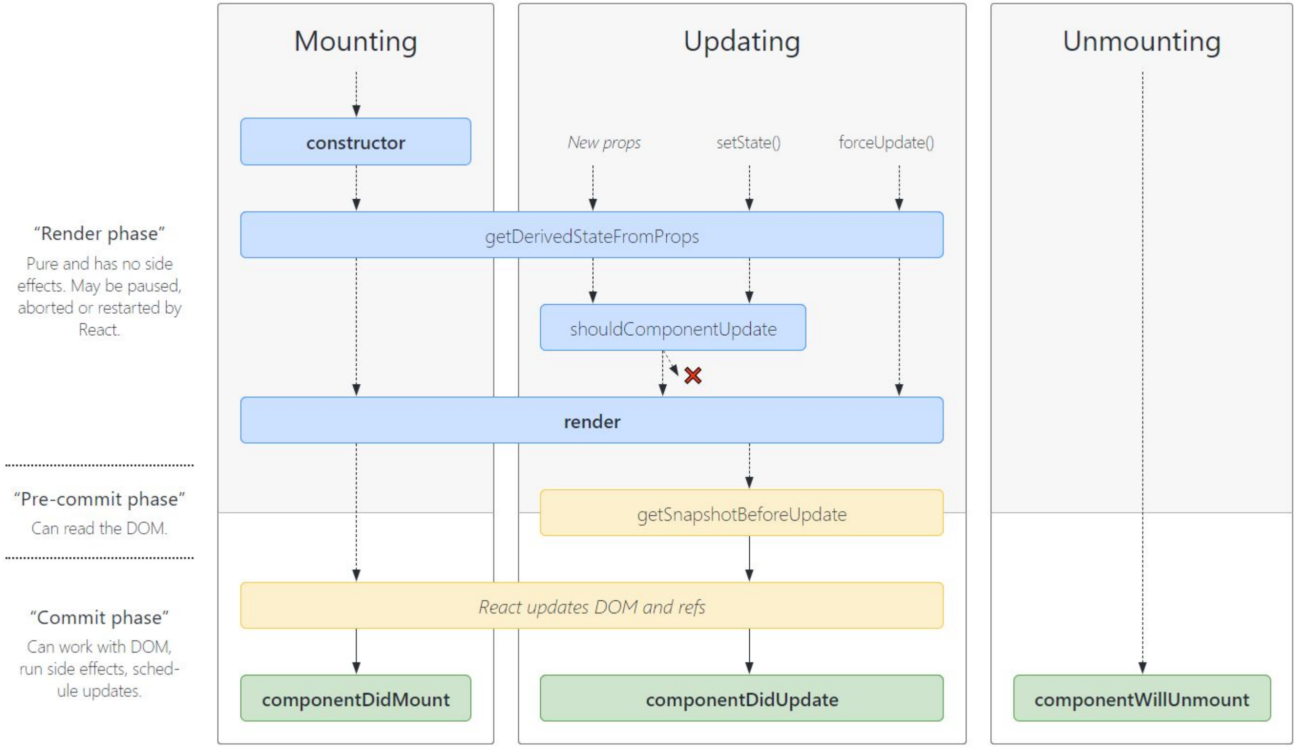
Error Handling - getDerivedStateFromError()

- This lifecycle is invoked after an error has been thrown by a descendant component.
- It receives the error that was thrown as a parameter and should return a value to update state.

Error Handling - componentDidCatch()

- This lifecycle is invoked after an error has been thrown by a descendant component. It receives two parameters:
  - error - The error that was thrown.
  - info - An object with a componentStack key containing information about which component threw the error

Life Cycle Overview



Thank You!

UCSC

Pasindu Marasinghe
ppm@ucsc.cmb.ac.lk