React 3: State & Lifecycle Methods

IN608: Intermediate Application Development Concepts

Kaiako: Tom Clark & Grayson Orr

Last Session's Content

- Components
  - Function
  - Class
- Props

Today's Content

- State
- Lifecycle methods
  - Mounting
  - Updating
  - Unmounting
- Data flow

State

State

- Consider the following component:

```
import React from 'react'

class Owner extends React.Component {
  render() {
    return (
      <div className='container'>
        <h1>My owner is {this.props.name}</h1>
      </div>
    )
  }
}

export default Owner
```

- We are going to add local state to this class component

State

- In the `render()`, replace `this.props.name` with `this.state.name`
- Add a class constructor which assigns the initial `this.state`
- What is state?
  - Contains data specific to the component
  - Data may change over time
  - State is user-defined & should be a JavaScript object
  - Never mutate `this.state` directly, i.e., `this.state.name = 'John Doe'`. Instead, use `setState()`
- All class components should always call the base constructor with `props`
- In `index.js`, remove the `name` prop

```
import React from 'react'

class Owner extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      name: 'Jane Doe',
    }
  }

  render() {
    return (
      <div className='container'>
        <h1>My owner is {this.state.name}</h1>
      </div>
    )
  }
}

export default Owner
```

State

- Create a new component file called `Clock.js`
- Later, we are going to add lifecycle methods to this class component

```
import React from 'react'

class Clock extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      date: new Date(),
    }
  }

  render() {
    return (
      <div className='container'>
        <h1>Current time: {this.state.date.toLocaleTimeString()}</h1>
      </div>
    )
  }
}

export default Clock
```

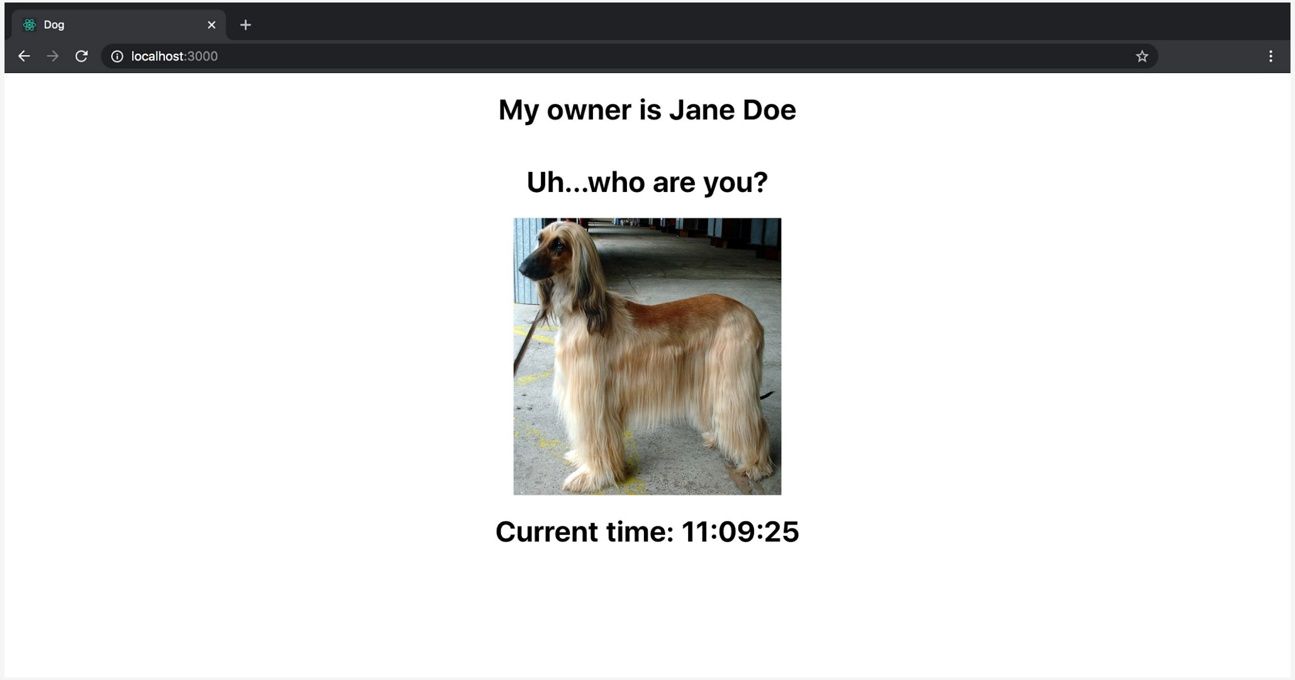
Lifecycle Methods

- In `index.js`

```
import React from 'react'
import ReactDOM from 'react-dom'
import './index.css'
import App from './components/App'
import Clock from './components/Clock'
import Owner from './components/Owner'

ReactDOM.render(
  <React.StrictMode>
    <Owner />
    <App />
    <Clock />
  </React.StrictMode>,
  document.getElementById('root')
)
```

State



My owner is Jane Doe

Uh...who are you?

Current time: 11:09:25

Lifecycle Methods

Lifecycle Methods

- Each component has several lifecycle methods which can be overridden to run at specific times during the process
- The component lifecycle is broken up into the following:
    - Mounting
    - Updating
    - Unmounting
    - Error handling
- We will only be concerned with the first three
- The mostly commonly used methods are in **bold**

Mounting

- When a component is being created & inserted into the DOM, the following methods are called:
  - **`constructor()`**
    - Called before the component is mounted
    - When implementing, `super(props)` should always be called first
    - May lead to bugs because `this.props` will be undefined
    - If you do not initialise state or do not bind methods, you do not have to implement `constructor()`
  - `static getDerivedStateFromProps()`
  - **`render()`**
    - The only required method for a class component
    - Does not modify component state
    - When invoked, returns the same result each time
  - **`componentDidMount()`**
    - Invoked immediately after a component is mount, i.e., inserted into the DOM tree

Updating

- When changes are made to props or state, the following methods are called:
    - `static getDerivedStateFromProps()`
    - `shouldComponentUpdate()`
    - **`render()`**
    - `getSnapshotBeforeUpdate()`
    - **`componentDidUpdate()`**
        - Invoked immediately after an update occurs
        - This is not called for the initial render when mounting a component

Unmounting

- When a component is removed from the DOM, the following method is called:
    - `componentWillMount()`
        - Invoked immediately before a component is unmounted & destroyed
        - Necessary cleanup is performed, i.e, cancelling network requests

Lifecycle Methods

- Declare a method called `tick()`
- `this.setState`
  - Enqueues changes to the component
  - Tells React that this component, i.e., `Clock` & its children need to be re-rendered with the update state, i.e., `new Date()`
  - A component does not always immediately update. It may batch or defer the update until later

```
import React from 'react'

class Clock extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      date: new Date(),
    }
  }

  tick() {
    this.setState({
      date: new Date(),
    })
  }

  render() {
    return (
      <div className='container'>
        <h1>Current time: {this.state.date.toLocaleTimeString()}</h1>
      </div>
    )
  }
}

export default Clock
```

Lifecycle Methods

- Declare `componentDidMount()` & `componentWillUnmount()` lifecycle methods

```
import React from 'react'

class Clock extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      date: new Date(),
    }
  }

  componentDidMount() {}

  componentWillUnmount() {}

  tick() {
    this.setState({
      date: new Date(),
    })
  }

  render() {
    return (
      <div className='container'>
        <h1>Current time: {this.state.date.toLocaleTimeString()}</h1>
      </div>
    )
  }
}

export default Clock
```

Lifecycle Methods

- You can add additional fields to a class component, i.e., `this.timerID`
- `setInterval(callback, delay)` - schedules repeated executions of a `callback`, i.e., `tick()` every `delay` milliseconds. Returns a `Timeout` object

```
import React from 'react'

class Clock extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      date: new Date(),
    }
  }

  componentDidMount() {
    this.timerID = setInterval(() => this.tick(), 1000)
  }

  componentWillUnmount() {}

  tick() {
    this.setState({
      date: new Date(),
    })
  }

  render() {
    return (
      <div className='container'>
        <h1>Current time: {this.state.date.toLocaleTimeString()}</h1>
      </div>
    )
  }
}

export default Clock
```

Lifecycle Methods

- `clearInterval(timeout)` - cancels a `Timeout` object created by `setInterval()`

```
import React from 'react'

class Clock extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      date: new Date(),
    }
  }

  componentDidMount() {
    this.timerID = setInterval(() => this.tick(), 1000)
  }

  componentWillUnmount() {
    clearInterval(this.timerID)
  }

  tick() {
    this.setState({
      date: new Date(),
    })
  }

  render() {
    return (
      <div className='container'>
        <h1>Current time: {this.state.date.toLocaleTimeString()}</h1>
      </div>
    )
  }
}

export default Clock
```

Lifecycle Methods

- Lets recap...
  - When `<Clock />` is passed to `ReactDOM.render()`, React calls the `Clock` component's constructor
  - Initialises local state by assigning an object to `this.setState`
  - React calls the `Clock` component's `render()` method
  - React updates the DOM to match the `Clock` component's render output
  - When the output is inserted into the DOM, React calls the `componentDidMount()` lifecycle method
  - `Clock` component asks the browser to setup a timer which calls the `tick()` method every 1000 milliseconds
  - React knows the state has changed & calls the `render()` method again. `this.setState.date` in the `render()` method will be different
  - If the `Clock` component is removed from the DOM, React calls the `componentWillUnmount()` lifecycle method so the timer is stopped

Data Flow

Data Flow

- Parent & child component can know if a component is stateful or stateless
- Components should not care if it defined as a function or class
- State is often called local or encapsulated...what does this mean? It is not accessible to any component other than the one that sets it
- A component may choose to pass its state to its child components as props, i.e., a child component would receive `date` in its props & would not know whether is came from
- This type of data flow is top-down meaning state is always owned by a specific component & data from that state can only affect components below them in the DOM tree