{codemotion}

React, The Inglorious Way

Matteo Antony Mistretta

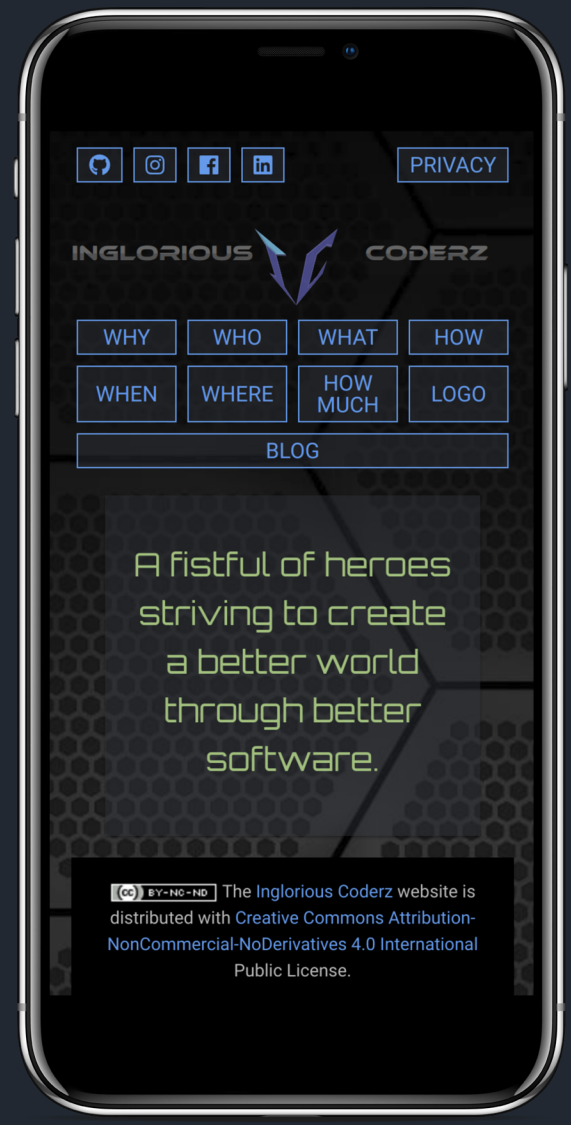
Inglorious Coderz

@antonymistretta

Why

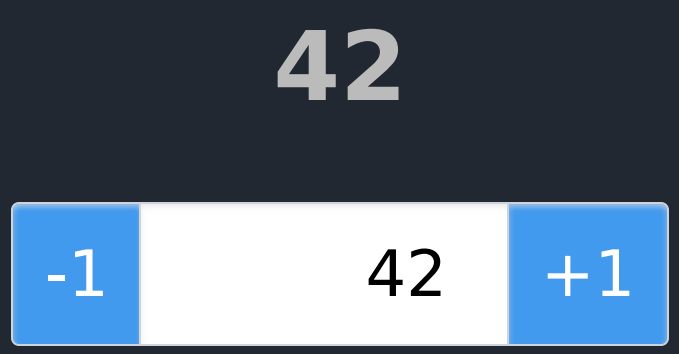
- React is evolving rapidly
- A few rules, lots of strategies
- Learning them makes us better coders

```
antony@ingloriouscoderz ~> whoami
```

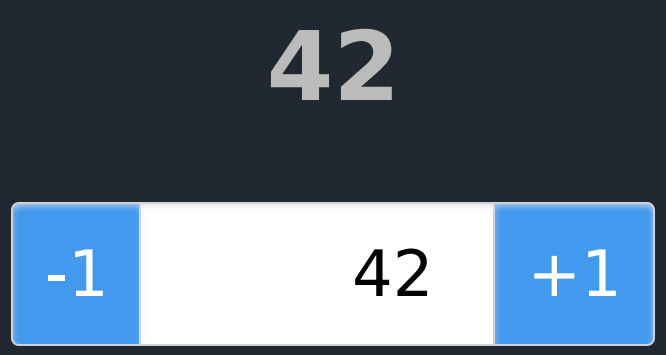


Agenda

- *Class Components*
- Container/Presentational
- Higher-Order Components
- Render Props
- Hooks



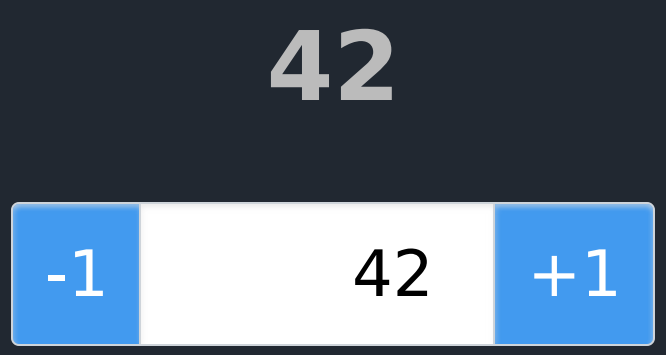

```
class Counter extends Component {
  constructor(props) {
    super(props)

    this.state = { count: props.initialCount }
    this.increment = this.increment.bind(this)
  }

  increment() {
    this.setState({ count: this.state.count + 1 })
  }

  decrement() {
    this.setState({ count: this.state.count - 1 })
  }

  render() {
    const { count } = this.state

    return (
      <div>
        <h1>{count}</h1>
        <div className="input-group">
          <button onClick={this.decrement.bind(this)}>-1</button>
          <input
            type="number"
            value={count}
            onChange={event => {
              this.setState({ count: parseInt(event.target.value) })
            }}
          />
          <button onClick={this.increment}>+1</button>
        </div>
      </div>
    )
  }
}

render(<Counter initialCount={42} />)
```



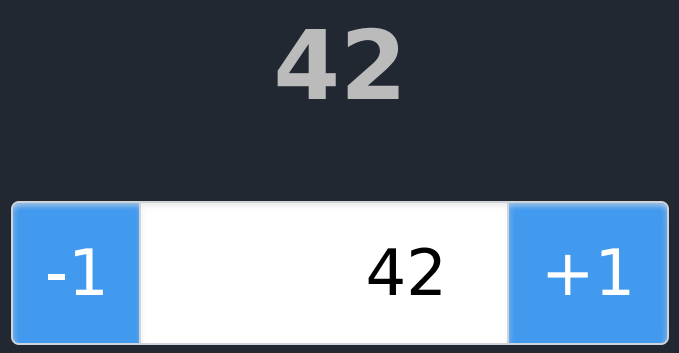

```jsx
class Counter extends PureComponent {
  state = { count: this.props.initialCount }

  increment = () => this.setState(({ count }) => ({ count: count + 1 }))
  decrement = () => this.setState(({ count }) => ({ count: count - 1 }))
  setCount = count => this.setState({ count })

  handleChange = event => this.setCount(parseInt(event.target.value))

  render() {
    const { count } = this.state

    return (
      <>
        <h1>{count}</h1>
        <div className="input-group">
          <button onClick={this.decrement}>-1</button>
          <input type="number" value={count} onChange={this.handleChange} />
          <button onClick={this.increment}>+1</button>
        </div>
      </>
    )
  }
}

render(<Counter initialCount={42} />)
```

Agenda

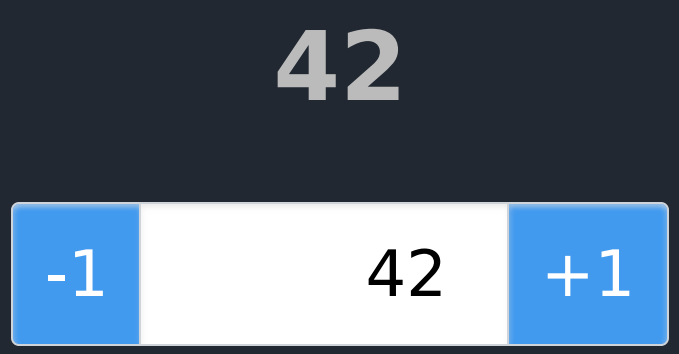
- Class Components
- *Container/Presentational*
- Higher-Order Components
- Render Props
- Hooks




```jsx
class Counter extends PureComponent {
  state = { count: this.props.initialCount }

  increment = () => this.setState(({ count }) => ({ count: count + 1 }))
  decrement = () => this.setState(({ count }) => ({ count: count - 1 }))
  setCount = count => this.setState({ count })

  handleChange = event => this.setCount(parseInt(event.target.value))

  render() {
    const { count } = this.state

    return (
      <>
        <h1>{count}</h1>
        <div className="input-group">
          <button onClick={this.decrement}>-1</button>
          <input type="number" value={count} onChange={this.handleChange} />
          <button onClick={this.increment}>+1</button>
        </div>
      </>
    )
  }
}

render(<Counter initialCount={42} />)
```



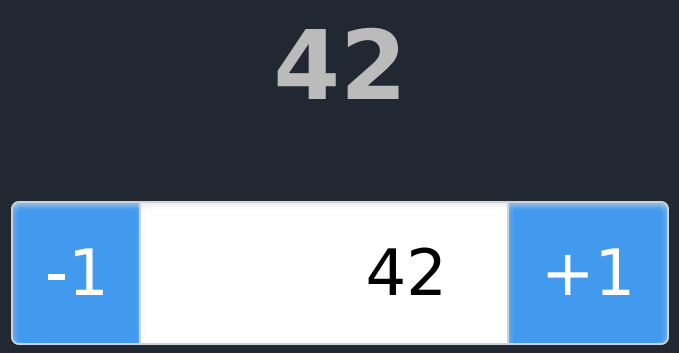

```jsx
class CounterContainer extends PureComponent {
  state = { count: this.props.initialCount }

  increment = () => this.setState(({ count }) => ({ count: count + 1 }))
  decrement = () => this.setState(({ count }) => ({ count: count - 1 }))
  setCount = count => this.setState({ count })

  handleChange = event => this.setCount(parseInt(event.target.value))

  render() {
    return (
      <Counter
        count={this.state.count}
        increment={this.increment}
        decrement={this.decrement}
        handleChange={this.handleChange}
      />
    )
  }
}

function Counter({ count, increment, decrement, handleChange }) {
  return (
    <>
      <h1>{count}</h1>
      <div className="input-group">
        <button onClick={decrement}>-1</button>
        <input type="number" value={count} onChange={handleChange} />
        <button onClick={increment}>+1</button>
      </div>
    </>
  )
}

render(<CounterContainer initialCount={42} />)
```



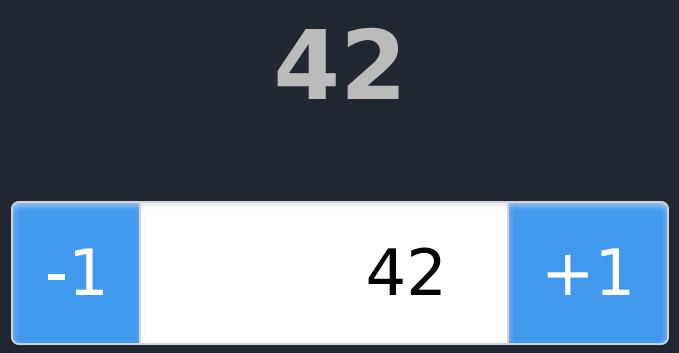

```jsx
class CounterContainer extends PureComponent {
  state = { count: this.props.initialCount }

  increment = () => this.setState(({ count }) => ({ count: count + 1 }))
  decrement = () => this.setState(({ count }) => ({ count: count - 1 }))
  setCount = count => this.setState({ count })

  handleChange = event => this.setCount(parseInt(event.target.value))

  render() {
    return Counter({
      count: this.state.count,
      increment: this.increment,
      decrement: this.decrement,
      handleChange: this.handleChange,
    })
  }
}

function Counter({ count, increment, decrement, handleChange }) {
  return (
    <>
      <h1>{count}</h1>
      <div className="input-group">
        <button onClick={decrement}>-1</button>
        <input type="number" value={count} onChange={handleChange} />
        <button onClick={increment}>+1</button>
      </div>
    </>
  )
}

render(<CounterContainer initialCount={42} />)
```

Agenda

- Class Components
- Container/Presentational
- *Higher-Order Components*
- Render Props
- Hooks

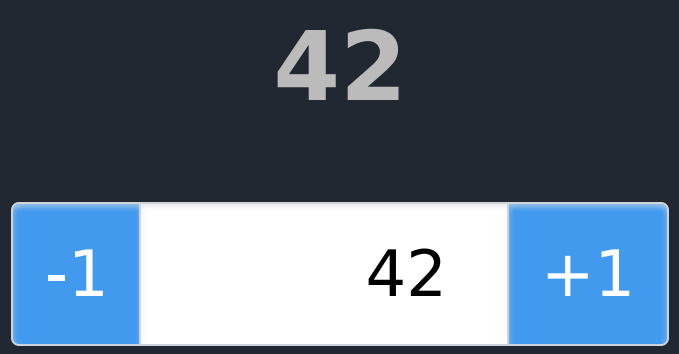



```jsx
class CounterContainer extends PureComponent {
  state = { count: this.props.initialCount }

  increment = () => this.setState(({ count }) => ({ count: count + 1 }))
  decrement = () => this.setState(({ count }) => ({ count: count - 1 }))
  setCount = count => this.setState({ count })

  handleChange = event => this.setCount(parseInt(event.target.value))

  render() {
    return (
      <Counter
        count={this.state.count}
        increment={this.increment}
        decrement={this.decrement}
        handleChange={this.handleChange}
      />
    )
  }
}

function Counter({ count, increment, decrement, handleChange }) {
  return (
    <>
      <h1>{count}</h1>
      <div className="input-group">
        <button onClick={decrement}>-1</button>
        <input type="number" value={count} onChange={handleChange} />
        <button onClick={increment}>+1</button>
      </div>
    </>
  )
}

render(<CounterContainer initialCount={42} />)
```




```
const withCounter = Enhanced =>
  class CounterContainer extends PureComponent {
    state = { count: this.props.initialCount }

    increment = () => this.setState(({ count }) => ({ count: count + 1 }))
    decrement = () => this.setState(({ count }) => ({ count: count - 1 }))
    setCount = count => this.setState({ count })

    handleChange = event => this.setCount(parseInt(event.target.value))

    render() {
      return (
        <Enhanced
          count={this.state.count}
          increment={this.increment}
          decrement={this.decrement}
          handleChange={this.handleChange}
        />
      )
    }
  }

Counter = withCounter(Counter)
```

```
function Counter({ count, increment, decrement, handleChange }) {
  return (
    <>
      <h1>{count}</h1>
      <div className="input-group">
        <button onClick={decrement}>-1</button>
        <input type="number" value={count} onChange={handleChange} />
        <button onClick={increment}>+1</button>
      </div>
    </>
  )
}

render(<Counter initialCount={42} />)
```




```
const enhance = compose(
  withState('count', 'setCount', ({ initialCount }) => initialCount),
  withHandlers({
    increment: ({ setCount }) => () => setCount(count => count + 1),
    decrement: ({ setCount }) => () => setCount(count => count - 1),
    handleChange: ({ setCount }) => event =>
      setCount(parseInt(event.target.value)),
  }),
  pure,
)

Counter = enhance(Counter)

function Counter({ count, increment, decrement, handleChange }) {
  return (
    <>
      <h1>{count}</h1>
      <div className="input-group">
        <button onClick={decrement}>-1</button>
        <input type="number" value={count} onChange={handleChange} />
        <button onClick={increment}>+1</button>
      </div>
    </>
  )
}

render(<Counter initialCount={42} />)
```

Agenda

- Class Components
- Container/Presentational
- Higher-Order Components
- *Render Props*
- Hooks

Hello world!

```
function Parent() {
  return (
    <Wrapper>
      <Child />
    </Wrapper>
  )
}

function Child() {
  return 'Hello world!'
}

function Wrapper({ children }) {
  return <h1>{children}</h1>
}

render(Parent)
```

Hello WORLD!

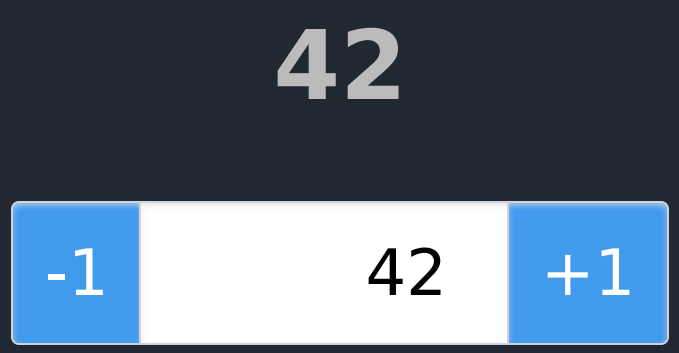
```
function Parent() {
  return <Wrapper Component={Child} who="world" />
}

function Child({ who }) {
  return `Hello ${who}!`
}

function Wrapper({ Component, who }) {
  const shoutedWho = who.toUpperCase()
  return (
    <h1>
      <Component who={shoutedWho} />
    </h1>
  )
}

render(Parent)
```

Hello WORLDz!

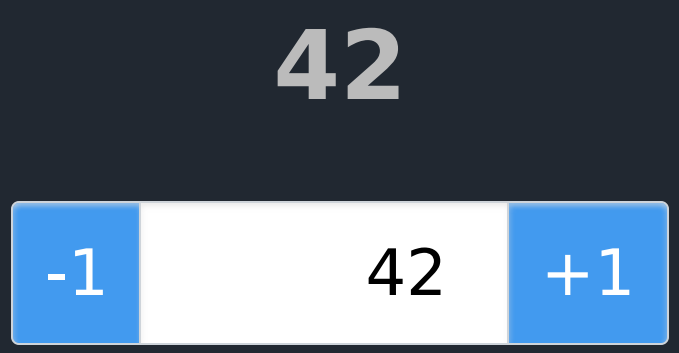
```
function Parent() {
  return <Wrapper render={who => <Child who={who + 'z'} />} who="world" />
}

function Child({ who }) {
  return `Hello ${who}!`
}

function Wrapper({ render, who }) {
  const shoutedWho = who.toUpperCase()
  return <h1>{render(shoutedWho)}</h1>
}

render(Parent)
```

Hello WORLDz!

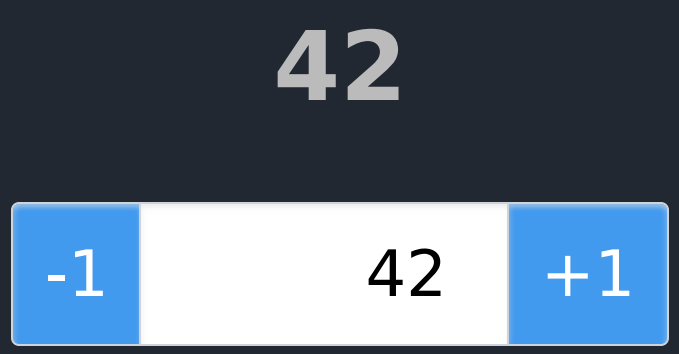
```
function Parent() {
  return <Wrapper who="world">{who => <Child who={who + 'z'} />}</Wrapper>
}

function Child({ who }) {
  return `Hello ${who}!`
}

function Wrapper({ children, who }) {
  const shoutedWho = who.toUpperCase()
  return <h1>{children(shoutedWho)}</h1>
}

render(Parent)
```




```jsx
class CounterContainer extends PureComponent {
  state = { count: this.props.initialCount }

  increment = () => this.setState(({ count }) => ({ count: count + 1 }))
  decrement = () => this.setState(({ count }) => ({ count: count - 1 }))
  setCount = count => this.setState({ count })

  handleChange = event => this.setCount(parseInt(event.target.value))

  render() {
    return (
      <Counter
        count={this.state.count}
        increment={this.increment}
        decrement={this.decrement}
        handleChange={this.handleChange}
      />
    )
  }
}

function Counter({ count, increment, decrement, handleChange }) {
  return (
    <>
      <h1>{count}</h1>
      <div className="input-group">
        <button onClick={decrement}>-1</button>
        <input type="number" value={count} onChange={handleChange} />
        <button onClick={increment}>+1</button>
      </div>
    </>
  )
}

render(<CounterContainer initialCount={42} />)
```




```jsx
class CounterContainer extends PureComponent {
  increment = () => this.setState(({ count }) => ({ count: count + 1 }))
  decrement = () => this.setState(({ count }) => ({ count: count - 1 }))
  setCount = count => this.setState({ count })

  handleChange = event => this.setCount(parseInt(event.target.value))

  state = {
    count: this.props.initialCount,
    increment: this.increment,
    decrement: this.decrement,
    handleChange: this.handleChange,
  }

  render() {
    return this.props.children(this.state)
  }
}

function Counter({ initialCount }) {
  return (
    <CounterContainer initialCount={initialCount}>
      {({ count, increment, decrement, handleChange }) => (
        <>
          <h1>{count}</h1>
          <div className="input-group">
            <button onClick={decrement}>-1</button>
            <input type="number" value={count} onChange={handleChange} />
            <button onClick={increment}>+1</button>
          </div>
        </>
      )}
    </CounterContainer>
  )
}

render(<Counter initialCount={42} />)
```

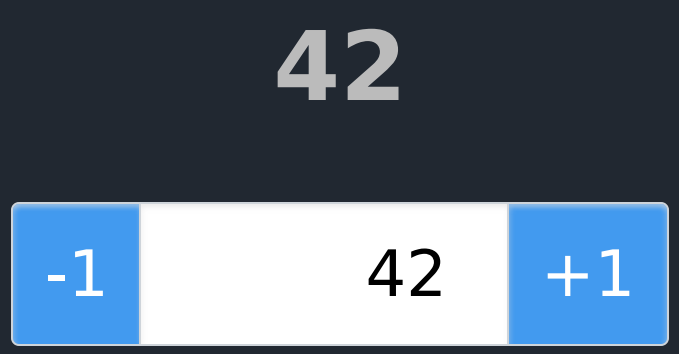



```jsx
class CounterContainer extends PureComponent {
  increment = () => this.setState(({ count }) => ({ count: count + 1 }))
  decrement = () => this.setState(({ count }) => ({ count: count - 1 }))
  setCount = count => this.setState({ count })

  handleChange = event => this.setCount(parseInt(event.target.value))

  state = {
    count: this.props.initialCount,
    increment: this.increment,
    decrement: this.decrement,
    handleChange: this.handleChange,
  }

  render() {
    return this.props.children(this.state)
  }
}

class Counter extends PureComponent {
  renderCounter = ({ count, increment, decrement, handleChange }) => (
    <>
      <h1>{count}</h1>
      <div className="input-group">
        <button onClick={decrement}>-1</button>
        <input type="number" value={count} onChange={handleChange} />
        <button onClick={increment}>+1</button>
      </div>
    </>
  )

  render() {
    return (
      <CounterContainer initialCount={this.props.initialCount}>
        {this.renderCounter}
      </CounterContainer>
    )
  }
}

render(<Counter initialCount={42} />)
```

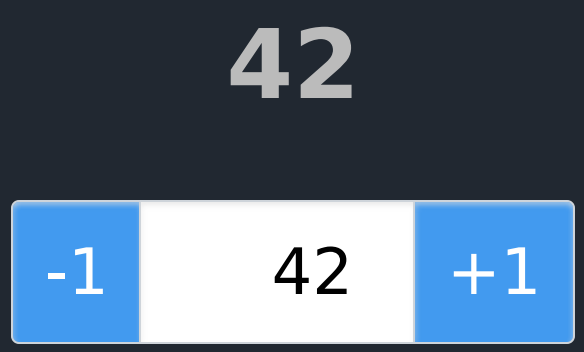
Agenda

- Class Components
- Container/Presentational
- Higher-Order Components
- Render Props
- *Hooks*

- They separate stateful logic
- They are composable functions
- They allow us to go fully functional
- They keep our component hierarchy flat

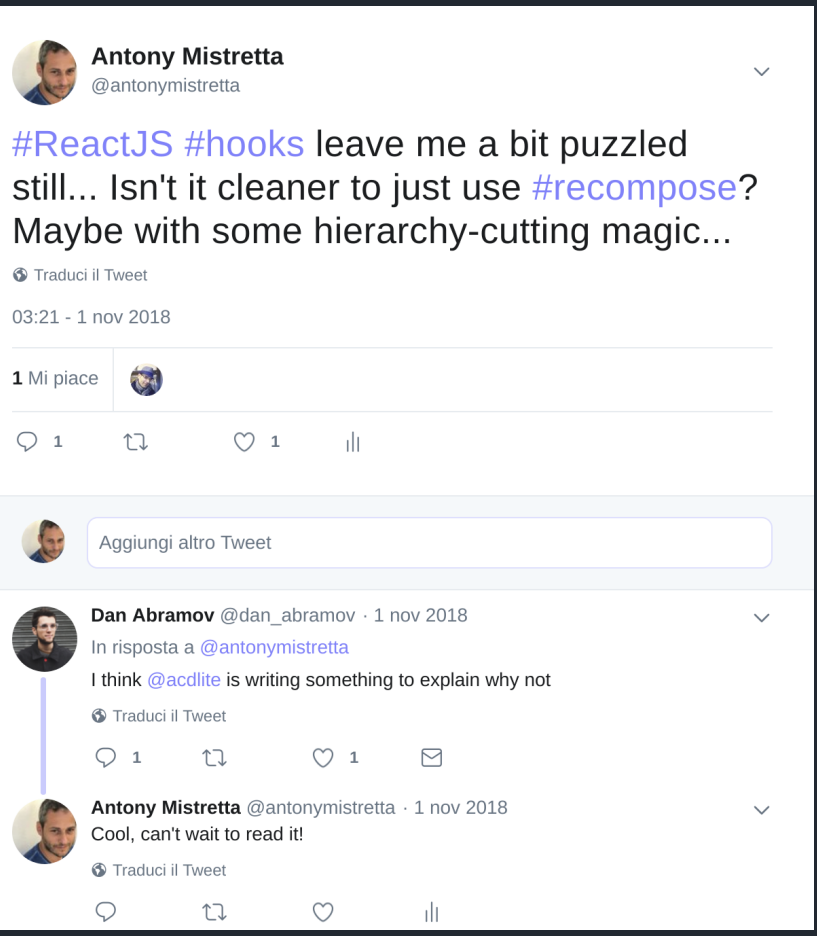



```
const enhance = compose(
  withState('count', 'setCount', ({ initialCount }) => initialCount),
  withHandlers({
    increment: ({ setCount }) => () => setCount(count => count + 1),
    decrement: ({ setCount }) => () => setCount(count => count - 1),
    handleChange: ({ setCount }) => event =>
      setCount(parseInt(event.target.value)),
  }),
  pure,
)

Counter = enhance(Counter)

function Counter({ count, increment, decrement, handleChange }) {
  return (
    <>
      <h1>{count}</h1>
      <div className="input-group">
        <button onClick={decrement}>-1</button>
        <input type="number" value={count} onChange={handleChange} />
        <button onClick={increment}>+1</button>
      </div>
    </>
  )
}

render(<Counter initialCount={42} />)
```




```jsx
function useCounter(initialCount) {
  const [count, setCount] = useState(initialCount)

  const increment = () => setCount(count + 1)
  const decrement = () => setCount(count - 1)
  const handleChange = event => setCount(parseInt(event.target.value))

  return { count, increment, decrement, handleChange }
}

Counter = memo(Counter)

function Counter({ initialCount }) {
  const { count, increment, decrement, handleChange } = useCounter(initialCount)

  return (
    <>
      <h1>{count}</h1>
      <div className="input-group">
        <button onClick={decrement}>-1</button>
        <input type="number" value={count} onChange={handleChange} />
        <button onClick={increment}>+1</button>
      </div>
    </>
  )
}

render(<Counter initialCount={42} />)
```

Which to use?


Antony Mistretta
@antonymistretta
#ReactJS #hooks leave me a bit puzzled still... Isn't it cleaner to just use #recompose? Maybe with some hierarchy-cutting magic...
Traduci il Tweet
03:21 - 1 nov 2018
1 Mi piace
1
1
Aggiungi altro Tweet
Dan Abramov @dan_abramov · 1 nov 2018
In risposta a @antonymistretta
I think @acdlite is writing something to explain why not
Traduci il Tweet
1
1
Antony Mistretta @antonymistretta · 1 nov 2018
Cool, can't wait to read it!
Traduci il Tweet

A Note from the Author (acdlite, Oct 25 2018):

Hi! I created Recompose about three years ago. About a year after that, I joined the React team. Today, we announced a proposal for *Hooks*. Hooks solves all the problems I attempted to address with Recompose three years ago, and more on top of that. I will be discontinuing active maintenance of this package (excluding perhaps bugfixes or patches for compatibility with future React releases), and recommending that people use Hooks instead. **Your existing code with Recompose will still work**, just don't expect any new features. Thank you so, so much to @wuct and @istarkov for their heroic work maintaining Recompose over the last few years.

gaearon commented on 14 Nov 2018 • edited

To clarify:

Andrew hasn't been working on new features in Recompose for two years. So declaring that he doesn't plan to add new features to it doesn't change anything in practice for existing users.

Andrew feels uneasy about recommending Recompose for new projects when Hooks solve a large subset of the same problems without introducing excessive tree nesting and similar issues. I agree the wording was perhaps too strong but he's the maintainer and he has the right to point out its flaws. He plans to write a longer article to explain what he meant because it seems like there's a lot of FUD going on.

If you're happy with Recompose and don't experience its downsides, you can keep on using it without any issues. New versions of Recompose will continue to be released together with React updates etc.

👍 55 😄 3 ❤️ 14

Hello WORLD!

```
function Hello({ who }) {
  return <h1>{`Hello ${who}!`}</h1>
}

const enhance = Enhanced => ({ who, ...props }) => {
  if (!who.length) return 'Name too short'

  const shoutedWho = who.toUpperCase()
  return <Enhanced {...props} who={shoutedWho} />
}

Hello = enhance(Hello)

render(<Hello who="world" />)
```

- world
- though
- ly
- tual
- issey

```
const SimpleList = ({ whos }) => (
  <ul style={styles.list}>
    {whos.map(who => (
      <li>{who}</li>
    ))}
  </ul>
)

const ComplexList = ({ renderEven, renderOdd, whos }) => (
  <ul style={styles.list}>
    {whos.map((who, index) => (index % 2 ? renderEven(who) : renderOdd(who)))}
  </ul>
)

const Parent = ({ whos, simple }) => {
  if (simple) return <SimpleList whos={whos} />

  return (
    <ComplexList
      renderEven={who => <li style={styles.even}>{'Even ' + who}</li>}
      renderOdd={who => <li style={styles.odd}>{'Odd ' + who}</li>}
      whos={whos}
    />
  )
}

render(
  <Parent whos={['world', 'though', 'ly', 'tual', 'issey']} simple={true} />,
)

const styles = {
  list: { textAlign: 'left' },
  odd: { color: 'grey' },
  even: { color: 'cornflowerblue' },
}
```

Which to use?

- Classes: getSnapshotBeforeUpdate / componentDidCatch
- HOCs: proxy / *before* render (change props, prevent renders)
- Render Props: mix logic / multiple sub-items / switch behavior
- Hooks: everything else

Thank you.

Questions?

html | pdf | source