React

Getting Started - React

A JavaScript library for building user interfaces

https://reactjs.org/docs/getting-started.html

Documentation

Components

```
import React from 'react'
import ReactDOM from 'react-dom'

class Hello extends React.Component {
  render () {
    return <div className='message-box'>
      Hello {this.props.name}
    </div>
  }
}

const el = document.body
ReactDOM.render(<Hello name='John' />, el)
```

Import multiple exports

```
import React, {Component} from 'react'
import ReactDOM from 'react-dom'

class Hello extends Component {
  ...
}
```

Proprieties

Use this.props to access properties passed to the component.

```
<Video fullscreen={true} autoplay={false} />

render () {
  this.props.fullscreen
  const { fullscreen, autoplay } = this.props
  ...
}
```

States

Use states (this.state) to manage dynamic data.

```
constructor(props) {
  super(props)
  this.state = { username: undefined }
}

this.setState({ username: 'rstacruz' })

render () {
  this.state.username
  const { username } = this.state
  ...
}
```

With Babel you can use proposal-class-fields and get rid of constructor

```
class Hello extends Component {
  state = { username: undefined };
  ...
}
```

Nesting

As of React v16.2.0, fragments can be used to return multiple children without adding extra wrapping nodes to the DOM.

```
class Info extends Component {
  render () {
    const { avatar, username } = this.props

    return <div>
      <UserAvatar src={avatar} />
      <UserProfile username={username} />
    </div>
  }
}
```

Nest components to separate concerns.

```
import React, {
  Component,
  Fragment
} from 'react'

class Info extends Component {
  render () {
    const { avatar, username } = this.props

    return (
      <Fragment>
        <UserAvatar src={avatar} />
        <UserProfile username={username} />
      </Fragment>
    )
  }
}
```

Children

Children are passed as the children property.

```
<AlertBox>
  <h1>You have pending notifications</h1>
</AlertBox>

class AlertBox extends Component {
  render () {
    return <div className='alert-box'>
      {this.props.children}
    </div>
  }
}
```

Setting default props

```
Hello.defaultProps = {
color: 'blue'
}
```

Setting default state

Set the default state in the constructor().

```
class Hello extends Component {
  constructor (props) {
    super(props)
    this.state = { visible: true }
  }
}
```

And without constructor using Babel with proposal-class-fields.

```
class Hello extends Component {
    state = { visible: true }
  }
}
```

Functional components

Functional components have no state. Also, their props are passed as the first parameter to a function.

```
function MyComponent ({ name }) {
  return <div className='message-box'>
    Hello {name}
  </div>
}
```

Pure components

Performance-optimized version of React.Component. Doesn't rerender if props/state hasn't changed.

```
import React, {PureComponent} from 'react'

class MessageBox extends PureComponent {
  ...
}
```

Component API

These methods and properties are available for Component instances.

```
this.forceUpdate()

this.setState({ ... })
this.setState(state => { ... })

this.state
this.props
```

Mounting

Set initial the state on constructor(). Add DOM event handlers, timers (etc) on componentDidMount(), then remove them on componentWillUnmount().

```
constructor (props)  // Before rendering #
componentWillMount()  // Don't use this #
render()  // Render #
componentDidMount() // After rendering (DOM available) #
componentWillUnmount()  // Before DOM removal #
componentDidCatch() // Catch errors (16+)
```

Updating

Called when parents change properties and .setState(). These are not called for initial renders.

```
// Use setState() here, but remember to compare props
componentDidUpdate (prevProps, prevState, snapshot)

// Skips render() if returns false
shouldComponentUpdate (newProps, newState)

// Render
render()

// Operate on the DOM here
componentDidUpdate (prevProps, prevState)
```

State hook

Hooks are a new addition in React 16.8.

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Declaring multiple state variables

```
function ExampleWithManyStates() {
  // Declare multiple state variables!
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
  // ...
}
```

Effect hook

If you're familiar with React class lifecycle methods, you can think of `useEffect` Hook as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` combined.

By default, React runs the effects after every render — including the first render.

```
import React, { useState, useEffect } from 'react';

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

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  }, [count]);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Building your own hoks

Define FriendStatus

Effects may also optionally specify how to “clean up” after them by returning a function.

```
import React, { useState, useEffect } from 'react';

function FriendStatus(props) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatus
    };
  }, [props.friend.id]);

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

UseFriendStatus

```
function FriendStatus(props) {
  const isOnline = useFriendStatus(props.friend.id);

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

Hooks API Reference

Basic hooks

```
useState(initialState)
useEffect(() => { … })
useContext(MyContext)  // value returned from React.createContext
```

Additional hooks

```
useReducer(reducer, initialArg, init)
useCallback(() => { … })
useMemo(() => { … })
useRef(initialValue)
useImperativeHandle(ref, () => { … })

// identical to useEffect, but it fires synchronously after all DOM muta
useLayoutEffect

// display a label for custom hooks in React DevTools
useDebugValue(value)
```

DOM nodes - References

Allow access to DOM nodes

```
class MyComponent extends Component {
  render () {
    return <div>
      <input ref={el => this.input = el} />
    </div>
  }

  componentDidMount () {
    this.input.focus()
  }
}
```

DOM Events

Pass functions to attributes like onChange.

```
class MyComponent extends Component {
  render () {
    <input type="text"
        value={this.state.value}
        onChange={event => this.onChange(event)} />
  }

  onChange (event) {
    this.setState({ value: event.target.value })
  }
```

Transferring props

Propagates src="..." down to the sub-component.

```
<VideoPlayer src="video.mp4" />

class VideoPlayer extends Component {
  render () {
    return <VideoEmbed {...this.props} />
  }
}
```

Top-level API

There are more, but these are most common.

```
React.createClass({ ... })
React.isValidElement(c)

ReactDOM.render(<Component />, domnode, [callback])
ReactDOM.unmountComponentAtNode(domnode)

ReactDOMServer.renderToString(<Component />)
ReactDOMServer.renderToStaticMarkup(<Component />)
```

JSX patterns

Style shorthand

```
const style = { height: 10 }

return <div style={style}></div>
return <div style={{ margin: 0, padding: 0 }}></div>
```

Inner HTML

```
function markdownify() { return "<p>...</p>"; }
<div dangerouslySetInnerHTML={{__html: markdownify()}} />
```

Lists

- Always supply a key property

```
class TodoList extends Component {
  render () {
    const { items } = this.props

    return <ul>
      {items.map(item =>
        <TodoItem item={item} key={item.key} />)}
    </ul>
  }
}
```

Conditionals

```
<Fragment>
  {showMyComponent
    ? <MyComponent />
    : <OtherComponent />}
</Fragment>
```

Short-circuit evaluation

```
<Fragment>
  {showPopup && <Popup />}
  ...
</Fragment>
```

Returning multiple elements

You can return multiple elements as arrays or fragments.

Arrays

```
render () {
  // Don't forget the keys!
  return [
    <li key="A">First item</li>,
    <li key="B">Second item</li>
  ]
}
```

Fragments

```
render () {

  // Fragments don't require keys!

  return (
    <Fragment>
      <li>First item</li>
      <li>Second item</li>
    </Fragment>
  )
}
```

Returning strings

You can return just a string

```
render() {
return 'Look ma, no spans!';
}
```

Errors

Catch errors via componentDidCatch. (React 16+)

```
class MyComponent extends Component {
  ...
  componentDidCatch (error, info) {
    this.setState({ error })
  }
}
```

Portals

This renders this.props.children into any location in the DOM.

```
render () {
  return React.createPortal(
    this.props.children,
    document.getElementById('menu')
  )
}
```

Hydratation

Use ReactDOM.hydrate instead of using ReactDOM.render if you're rendering over the output of ReactDOMServer.

```
const el = document.getElementById('app')
ReactDOM.hydrate(<App />, el)
```

Property validation

PropTypes

```
import PropTypes from 'prop-types'

any // Anything
Basic
string
number
func  // Function
bool  // True or false
Enum
oneOf(any)  // Enum types
```

```
oneOfType(type array) // Union
Array
array
arrayOf(…)
Object
object
objectOf(…) // Object with values of a certain type
instanceOf(…) // Instance of a class
shape(…)
Elements
element // React element
node  // DOM node
Required
(···).isRequired  // Required
```

Basic types

```
MyComponent.propTypes = {
  email:      PropTypes.string,
  seats:      PropTypes.number,
  callback:   PropTypes.func,
  isClosed:   PropTypes.bool,
  any:        PropTypes.any
}
```

Required types

```
MyCo.propTypes = {
  name:  PropTypes.string.isRequired
}
```

Elements

```
MyCo.propTypes = {
  // React element
  element: PropTypes.element,

  // num, string, element, or an array of those
  node: PropTypes.node
}
```

Enumerable (oneOf)

```
MyCo.propTypes = {
  direction: PropTypes.oneOf([
    'left', 'right'
  ])
}
```

Arrays and objects

- Use .array[Of], .object[Of], .instanceOf, .shape.

```
MyCo.propTypes = {
  list: PropTypes.array,
  ages: PropTypes.arrayOf(PropTypes.number),
  user: PropTypes.object,
  user: PropTypes.objectOf(PropTypes.number),
  message: PropTypes.instanceOf(Message)
}
MyCo.propTypes = {
  user: PropTypes.shape({
    name: PropTypes.string,
    age:  PropTypes.number
  })
}
```

Custom validation

```
MyCo.propTypes = {
  customProp: (props, key, componentName) => {
    if (!/matchme/.test(props[key])) {
      return new Error('Validation failed!')
    }
  }
}
```