FullStack.Cafe - Kill Your Tech Interview

Q1: How does React *work*? ☆

Topics: React

Answer:

React creates a virtual DOM. When state changes in a component it firstly runs a "diffing" algorithm, which identifies what has changed in the virtual DOM. The second step is reconciliation, where it updates the DOM with the results of diff.

Q2: How would you write an *inline* style in React? ☆

Topics: React

Answer:

For example:

```
<div style={{ height: 10 }}>
```

Q3: What are the major *features* of ReactJS? ☆

Topics: React

Answer:

The major features of ReactJS are as follows,

- It uses **VirtualDOM** instead RealDOM considering that RealDOM manipulations are expensive.
- Supports **server-side rendering**
- Follows **Unidirectional** data flow or data binding
- Uses **reusable/composable** UI components to develop the view

Q4: What are `props` in React? ☆

Topics: React

Answer:

Props are inputs to a React component. They are single values or objects containing a set of values that are passed to React Components on creation using a naming convention similar to HTML-tag attributes. i.e, *They are data passed down from a parent component to a child component.*

The primary purpose of props in React is to provide following component functionality:

1. Pass custom data to your React component.

2. Trigger `state` changes.
3. Use via `this.props.reactProp` inside component's `render()` method.

For example, let us create an element with reactProp property,

```
<Element reactProp = "1" />
```

This `reactProp` (or whatever you came up with) name then becomes a property attached to React's native props object which originally already exists on all components created using React library.

```
props.reactProp;
```

Q5: What is the use of `refs`? ☆

Topics: React

Answer:

Refs provide a way to access DOM nodes or React elements created in the render method. They should be avoided in most cases, however, they can be useful when we need direct access to DOM element or an instance of a component.

There are a few good use cases for refs:

- Managing focus, text selection, or media playback.
- Triggering imperative animations.
- Integrating with third-party DOM libraries.

Refs are created using `React.createRef()` and attached to React elements via the `ref` attribute. Refs are commonly assigned to an instance property when a component is constructed so they can be referenced throughout the component.

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.myRef = React.createRef();  }
  render() {
    return <div ref={this.myRef} />;  }
}
```

Q6: What is *Context API* in ReactJS? ☆

Topics: React

Answer:

Context provides a way to pass data through the component tree without having to pass props down manually at every level. Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language. Using context, we can avoid passing props through intermediate elements.

```jsx
// Context lets us pass a value deep into the component tree
// without explicitly threading it through every component.
// Create a context for the current theme (with "light" as the default).
const ThemeContext = React.createContext('light');

class App extends React.Component {
  render() {
    // Use a Provider to pass the current theme to the tree below.
    // Any component can read it, no matter how deep it is.
    // In this example, we're passing "dark" as the current value.
    return (
      <ThemeContext.Provider value="dark">
        <Toolbar />
      </ThemeContext.Provider>
    );
  }
}

// A component in the middle doesn't have to
// pass the theme down explicitly anymore.
function Toolbar() {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

class ThemedButton extends React.Component {
  // Assign a contextType to read the current theme context.
  // React will find the closest theme Provider above and use its value.
  // In this example, the current theme is "dark".
  static contextType = ThemeContext;
  render() {
    return <Button theme={this.context} />;
  }
}
```

Q7: What are the *advantages* of ReactJS? ☆

Topics: React

Answer:

Below are the advantages of ReactJS:

1. Increases the application's performance with Virtual DOM
2. JSX makes code is easy to read and write
3. It renders both on client and server side
4. Easy to integrate with other frameworks (Angular, BackboneJS) since it is only a view library
5. Easy to write UI Test cases and integration with tools such as JEST.

Q8: What are React Hooks? ☆

Topics: React React Hooks

Answer:

Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class. With Hooks, you can extract stateful logic from a component so it can be tested independently and reused.

Hooks allow you to reuse stateful logic without changing your component hierarchy. This makes it easy to share Hooks among many components or with the community.

Q9: What are the *advantages* of using React? ☆☆

Topics: React

Answer:

- It is easy to know how a component is rendered, you just need to look at the render function.
- JSX makes it easy to read the code of your components. It is also really easy to see the layout, or how components are plugged/combined with each other.
- You can render React on the server-side. This enables improves SEO and performance.
- It is easy to test.
- You can use React with any framework (Backbone.js, Angular.js) as it is only a view layer.

Q10: What is the difference between a *Presentational component* and a *Container component*? ☆☆

Topics: React

Answer:

- **Presentational components** are concerned with *how things look*. They generally receive data and callbacks exclusively via props. These components rarely have their own state, but when they do it generally concerns UI state, as opposed to data state.

- **Container components** are more concerned with *how things work*. These components provide the data and behavior to presentational or other container components. They call Flux actions and provide these as callbacks to the presentational components. They are also often stateful as they serve as data sources.

Q11: What are the differences between a *Class component* and *Functional component*? ☆☆

Topics: React

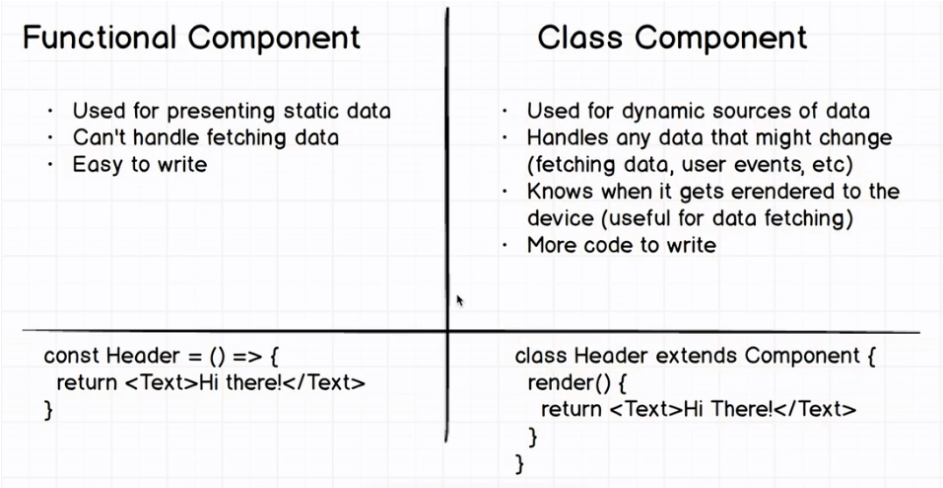
Answer:

Class Components

- Class-based Components uses ES6 class syntax. It can make use of the lifecycle methods.
- Class components extend from React.Component.
- In here you have to use this keyword to access the props and functions that you declare inside the class components.

Functional Components

- Functional Components are simpler comparing to class-based functions.
- Functional Components mainly focuses on the UI of the application, not on the behavior.
- To be more precise these are basically render function in the class component.
- Functional Components can have state and mimic lifecycle events using Reach Hooks



Q12: What is the difference between `state` and `props`? ☆☆

Topics: React

Answer:

- The **state** is a data structure that starts with a default value when a Component mounts. It may be mutated across time, mostly as a result of user events.
- **Props** (short for properties) are a Component's configuration. They are received from above and immutable as far as the Component receiving them is concerned. A Component cannot change its props, but it is responsible for putting together the props of its child Components. Props do not have to just be data - callback functions may be passed in as props.

Q13: What are `refs` used for in React? ☆☆

Topics: React

Answer:

Refs are an escape hatch which allow you to get direct access to a DOM element or an instance of a component. In order to use them you add a ref attribute to your component whose value is a callback function which will receive the underlying DOM element or the mounted instance of the component as its first argument.

```
class UnControlledForm extends Component {
  handleSubmit = () => {
    console.log("Input Value: ", this.input.value)
  }
  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          ref={(input) => this.input = input} />
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

Above notice that our input field has a ref attribute whose value is a function. That function receives the actual DOM element of input which we then put on the instance in order to have access to it inside of the handleSubmit

function.

It’s often misconstrued that you need to use a class component in order to use refs, but refs can also be used with functional components by leveraging closures in JavaScript.

```
function CustomForm ({handleSubmit}) {
  let inputElement
  return (
    <form onSubmit={() => handleSubmit(inputElement.value)}>
      <input
        type='text'
        ref={(input) => inputElement = input} />
      <button type='submit'>Submit</button>
    </form>
  )
}
```

Q14: When rendering a list what is a `key` and what is it's purpose? ☆☆

Topics: React

Answer:

Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity. The best way to pick a key is to use a string that uniquely identifies a list item among its siblings.

```
render () {
  return (
    <ul>
      {this.state.todoItems.map(({task, uid}) => {
        return <li key={uid}>{task}</li>
      })}
    </ul>
  )
}
```

Most often you would use IDs from your data as keys. When you don't have stable IDs for rendered items, you may use the item index as a key as a last resort. It is not recommend to use indexes for keys if the items can reorder, as that would be slow.

Q15: What does it mean for a component to be *mounted* in React? ☆☆

Topics: React

Answer:

It has a corresponding element created in the DOM and is connected to that.

Q16: What's the difference between a *Controlled* component and an *Uncontrolled* one in React? ☆☆

Topics: React

Answer:

This relates to stateful DOM components (form elements) and the React docs explain the difference:

- A Controlled Component is one that takes its current value through `props` and notifies changes through callbacks like `onChange`. A parent component "controls" it by handling the callback and managing its own state and passing the new values as props to the controlled component. You could also call this a "dumb component".
- A Uncontrolled Component is one that stores its own state internally, and you query the DOM using a `ref` to find its current value when you need it. This is a bit more like traditional HTML.

Most native React form components support both controlled and uncontrolled usage:

```
// Controlled:
<input type="text" value={value} onChange={handleChange} />

// Uncontrolled:
<input type="text" defaultValue="foo" ref={inputRef} />
// Use `inputRef.current.value` to read the current value of <input>
```

In most (or all) cases you should use controlled components.

Q17: What are *Controlled components* in ReactJS? ☆☆

Topics: React

Answer:

A **Controlled Component** is one that takes its current value through `props` and notifies changes through callbacks like `onChange`. A parent component "controls" it by handling the callback and managing its own state and passing the new values as props to the controlled component. You could also call this a "dumb component".

```
// Controlled:
<input type="text" value={value} onChange={handleChange} />
```

Q18: What is *Reconciliation* in ReactJS? ☆☆

Topics: React

Answer:

When a component's props or state change, React decides whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one. When they are not equal, React will update the DOM. This process is called **reconciliation**.

Q19: What are *Fragments* in React? ☆☆

Topics: React

Answer:

It's common pattern in React which is used for a component to *return multiple elements*. **Fragments** let you group a list of children without adding extra nodes to the DOM.

```
render() {
  return (
    <React.Fragment>
      <ChildA />
      <ChildB />
      <ChildC />
    </React.Fragment>
  );
}
```

There is also a **shorter syntax**:

```
render() {
    return (
      <>
         <ChildA />
         <ChildB />
         <ChildC />
      </>
    );
  }
```

Q20: What is the difference between *Component* and *Container* in Redux? ☆☆

Topics: React Redux

Answer:

- **Component** is part of the React API. A Component is a class or function that describes part of a React UI.
- **Container** is an informal term for a React component that is connected to a redux store. Containers receive Redux state updates and dispatch actions, and they usually don't render DOM elements; they delegate rendering to presentational child components.