React Component in details

Lesson 07

Capgemini

Lesson Objectives

At the end of this module on React fundamentals you will be able to:

- Higher Order Components
- Passing unknown Props
- Validating Props
- Using References
- React Context API
- Updated LifeCycle hooks (16.3)
- Best practices for React Projects

Nested Components

Components can be nested inside other components.

Components are self-contained, and thus enable to nest multiple components with one another.

```
var OuterComponent = React.createClass({ {/*Top level Component*/}
        render:function(){
                return(<div><InnerComponent/></div>)
        }
});
var InnerComponent = React.createClass({
        render:function(){
                return (<div>Inner Component</div>)
        }
});
ReactDOM.render(<OuterComponent/>,document.getElementById('app'));
```

React Component which has a child component is called as Top level component.

Demo

Nested-Components

Higher Order Components

- A higher-order component (HOC) is an advanced technique in React for reusing component logic. HOCs are not part of the React API, they are a pattern that emerges from React's compositional nature
- A higher-order component is a function that ***takes a component and returns a new component***.(also are referred as functional programming methodology).

- **Advantages:**
  - Easy to test
  - Pass through props unrelated to the HOC
  - when you need a repeating pattern
- **Disadvantages:**
  - Don't use it inside render method
  - Do not mutate Original component
  - Don't overuse this pattern

Demo

Higher order components

React-higherorder-2019

Working with props (Properties)

The props parameter is a JavaScript object (data & event handlers) passed from a parent element to a child element.

Props are supplied as attributes:

- If the value of the attribute is **JavaScript expression** it must be enclosed in curly Brackets ({}).
- If it is a **string literal** it must be enclosed with in double quotes ("").

Props can be accessed via props property inside a component.

Props are considered to be immutable; it stores read-only data that is passed from the parent. It belongs to the parent and cannot be changed by its children.

getDefaultProps() specifies property values to use, if they are not explicitly supplied.

Parent can read its children by accessing the special this.props.children prop.

Demo

Using-Props

Create-React-props

Passing Unknown Props

- When we have a hierarchy of components ie., nested components , where we get a scenario to pass props from parent component to child components without modifying the props. Generally what we code shown below.

```
const ParentComponent = (props) =>
{
return ( <ChildComponent prop1={props.prop1} prop2={props.prop2} /> )
}
```

- But there is an easy and alternate approach to do this is by using **Spread operator .**

```
const ParentComponent = (props) => {
return (
<ChildComponent {...props} />
 ) }
```

- This syntax is much easier, less error prone, and it allows flexibility, since you don’t need to change the props names or add props in the intermediate component when you change them

JSX Spread Attributes

The ... operator is called as spread operator.

Using JSX Spread Attributes, we can construct the props before creating the components and pass them later to the components.

```
var props = {};
  props.foo = x;
  props.bar = y;
  var component = <Component {...props} />;
```

The properties of the object that passed in are copied onto the component's props.

We can transfer it multiple times, combine it with other attributes and override the value.

```
var props = { foo: 'default' };
  var component = <Component {...props} foo={'override'}
/>;
  console.log(component.props.foo); // 'override'
```

Demo

React-Spread-attr-2019

Prop Validation

React offers a great suite of validators for checking the props set for a component are as expected.

React.PropTypes exports a range of validators that can be used to make sure the data you receive is valid.

React supports validation of existence, data type or a custom condition.

Using the following prop types we can validate whether a prop:

- is required
- contains a primitive type
- contains something renderable (a node)
- is a React Element
- contains one of several defined types
- is an array containing only items of a specified type
- contains an instanceof a class
- contains an object that has a specific shape

For performance reasons propTypes is only checked in development mode.

eg:

```
MyComponent.propTypes = {
    children: PropTypes.element.isRequired
};
```

Demo

Using-Props

create-react-props-validation
react-proptypes

Using Refs

- Refs are introduced in React 16.3.
- Refs provide a way to access DOM nodes or React elements created in the render method.
- Generally, refs can be used only when not able to something through **state** and **props**.
- **When to use refs:**
  - Integrating with third-party DOM libraries
  - Managing focus, handling text selections or media playback behavior.
  - Triggering imperative animations
- The ref is first set after the first render(), but before componentDidMount().
- **Disadvantages of Ref:**
  - It hinders in optimized working of Babel inline plugin
  - Once state changes, you re-render **all** the components of your UI that depend on that state.

Demo

react-refs-2019

Accessing User Input With refs

React provides two standard ways to grab values from <form> elements. The first method is to implement what are called controlled components The second is to use React's ref property.

Ex:

```
<input type="text" ref={input => this.fullName = input}/>

<input type="number" ref={cashMoney => this.amount =
cashMoney} />
```

We can also use for radio box, check box.

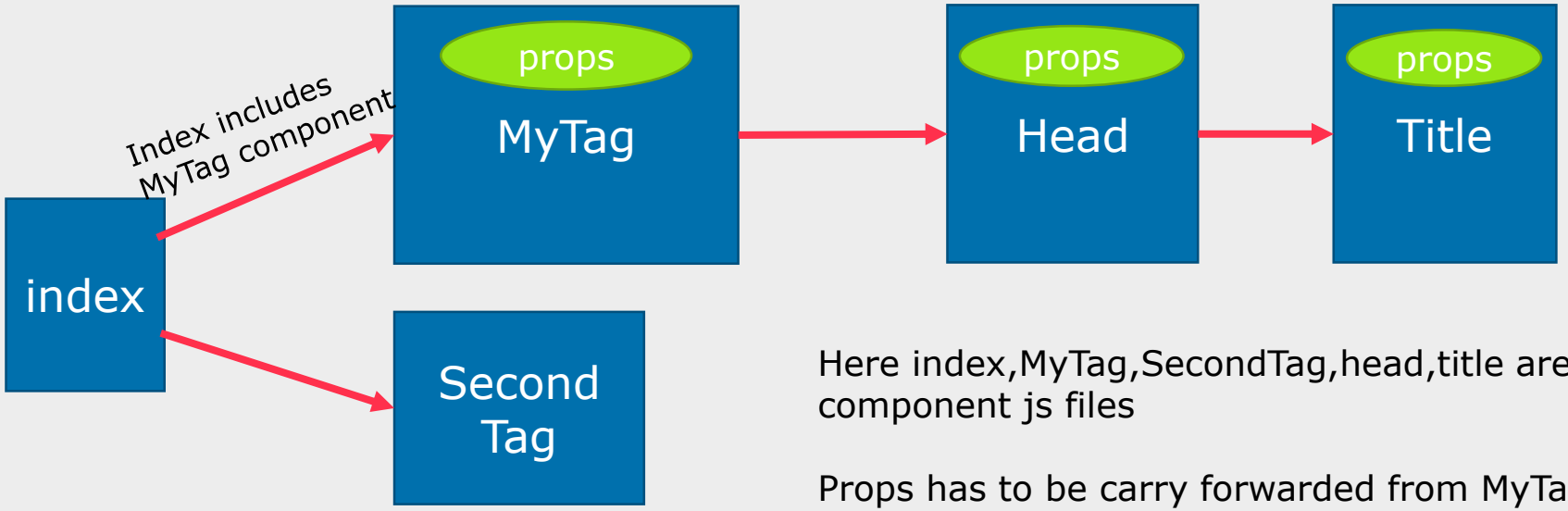
The primary value of using refs over controlled component is that, in most cases, you will write less code.

Demo

Create-react-forms-inputrefs

Context API

- Context provides a way to pass data through the component tree without having to pass props down manually at every level.
- This is called as props-drilling



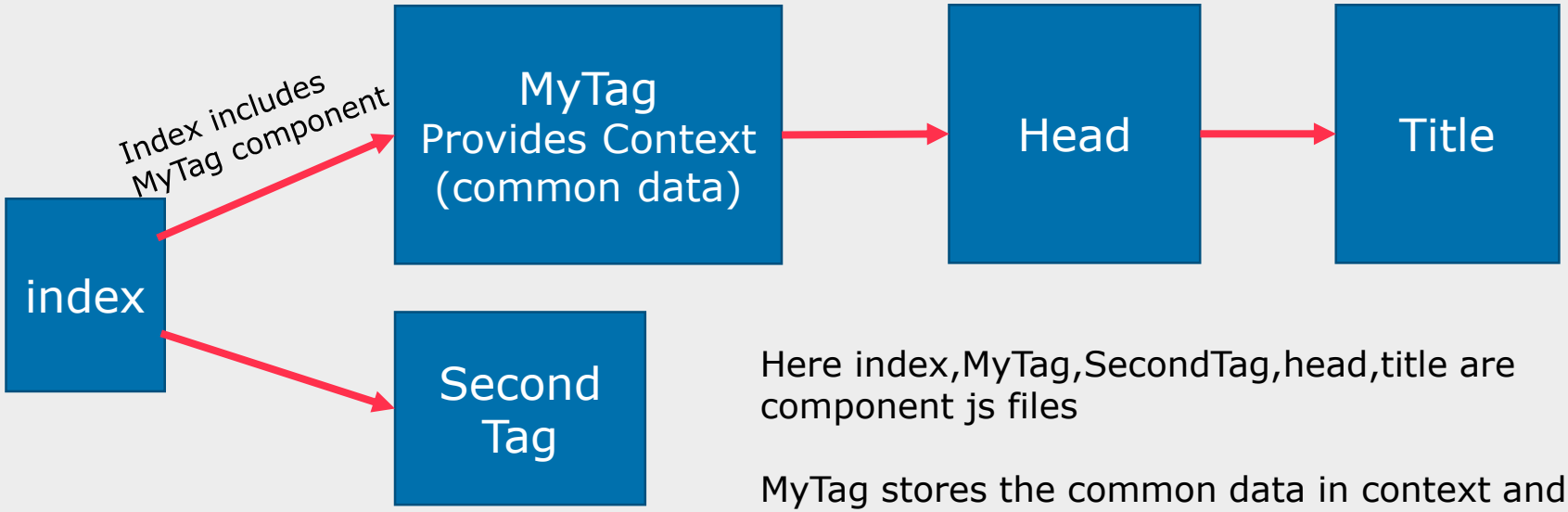
Here index,MyTag,SecondTag,head,title are component js files

Props has to be carry forwarded from MyTag to Head, Head to Title(props-drilling)

- React's context API gives you a solution for props –drilling , instead of passing down the props explicitly down to each component you can store the data needed by every Component in React's context and pass them to all other components implicitly.
- If a component needs access to the context, it can consume it on demand.

Context continued…

- After usage of react's Context Api



Here index,MyTag,SecondTag,head,title are component js files

MyTag stores the common data in context and are utilized by head and Title components

When to Use Context

- Context can be used when data that can be considered "global" for a tree of React components. such as the current authenticated user, theme(css), company details or preferred language.

Context Continued….

- React Context has 2 components
  - Consumer
  - Provider
- To access these 2 components, we have to create Context object
  - createContext()
    - const GradeContext = React.createContext(‘grades’);

Consumer :

```
<ThemeContext.Provider value={'green'}>
 <D />
</ThemeContext.Provider>
```

Provider:

```
<ThemeContext.Consumer>
{coloredTheme =>
  <div style={{ color: coloredTheme }}>
    Hello World
  </div>  }
</ThemeContext.Consumer>
```

Demo

Demos of

Create-react-context2

React Updated LifeCycle hooks

- ReactJs v16.3 introduced significant changes in component lifecycle. In React 16.3 few lifecycle methods have been deprecated. Those methods are prefixed by UNSAFE_.
- Methods like componentWillMount, componentWillReceiveProps and
- componentWillUpdate were heavily misused because the current instance this was available and is easy to misuse.
- **latest React component lifecycle**
  - The React component which extends React.Component goes through the following phases:
    - Mounting
    - Updating
    - Unmounting

React Updated LifeCycle hooks contd.

- **latest React component lifecycle**
  - The React component which extends React.Component goes through the following phases:
    - Mounting
    - Updating
    - Unmounting

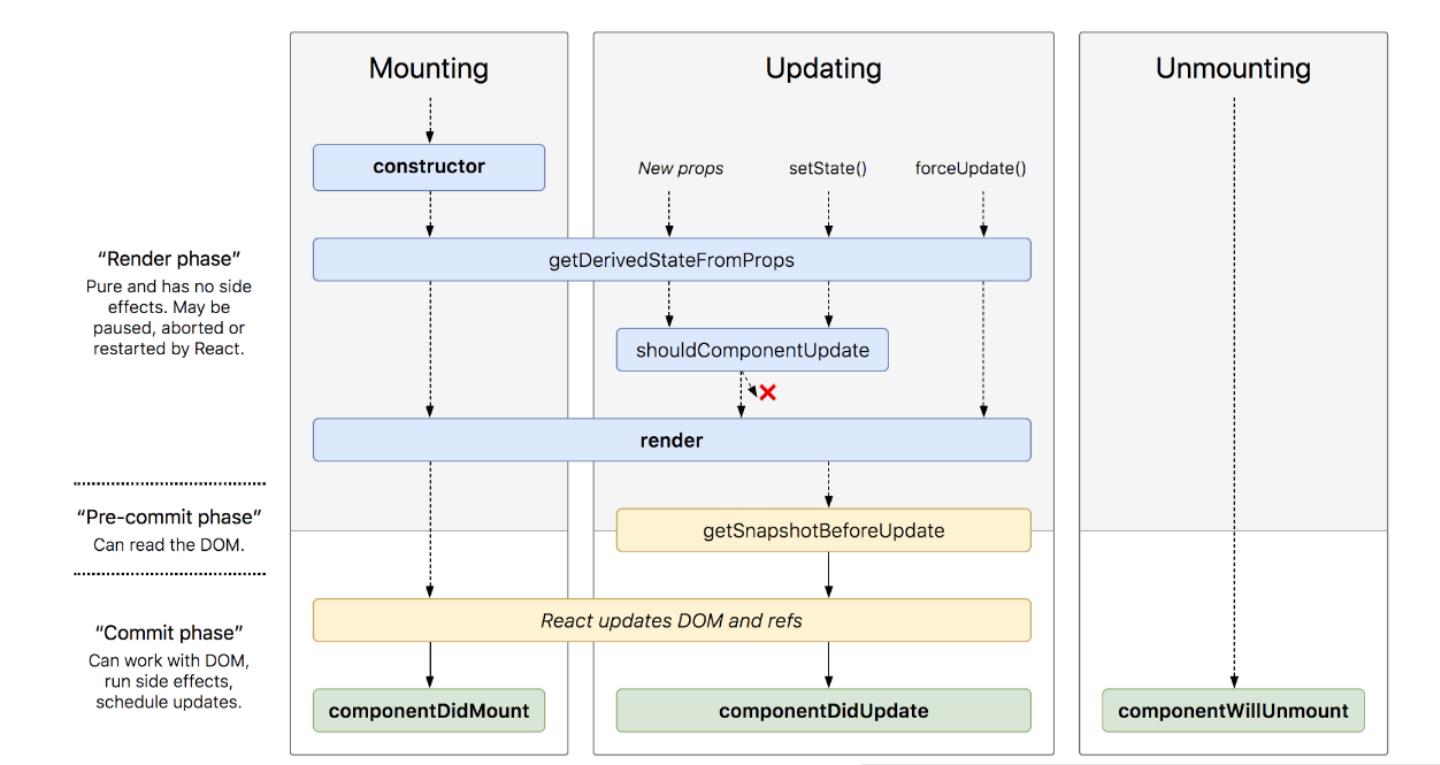


Image taken from https://blog.bitsrc.io/

Lifecycle hooks contd.

Mounting :

constructor

The first thing that gets called is your component constructor, *if* your component is a class component. This does not apply to functional components.

getDerivedStateFromProps

Last Method in Mounting phase is getDerivedStateFromProps , which is called before render method and is used for set state according to the props. whenever the props are changed the state has to be kept in sync.

This method is a safer replacement of componentWillReceiveProps

```
static getDerivedStateFromProps(props, state) {
return {
      blocks: createBlocks(props.numberOfBlocks)
      };
}
```

Life cycle hooks contd.

render

Rendering does all the work. It returns the JSX of your actual component.

componentDidMount

This is the hook method which is invoked immediately after the component **did** mount on the browser DOM. If we need to load any data we can do here. API calls should be made in this method.

2. Updating

This Phase starts whenever React Component needs to be updated with the changes. They can be updated in 2 ways.

1. sending new props from parents
2. updating the current state

static getDerivedStateFromProps

This method behaves exactly as defined above in mounting phase

shouldComponentUpdate

This method tells React that when the component is being updated, it should re-render or ignore rendering. The method returns true or false based on which component is re-rendered or ignored.

Render:

Again render method is called to to display component in browser

Life cycle hooks contd.

getSnapshotBeforeUpdate

This method gets called after the render created the React element and before it is actually updated from virtual DOM to actual DOM. This phase is known as pre-commit phase.

componentDidUpdate

is executed when the newly updated component has been updated in the DOM. This method is used to re-trigger the third party libraries used, and to make sure these libraries also update and reload themselves.

3) Unmounting

In this phase, the component is not needed and the component will get unmounted from the DOM. Below method is called

componentWillUnmount :

This method is the last method in the lifecycle. This is executed just before the component gets removed from the DOM.

Best Practices in React Js

- Don’t duplicate source of truth—props in initial state is an anti pattern

- The state should be avoided as much as possible. It is a good practice to centralize the state and pass it down the component tree as props. Use Flux pattern for Handling the state.

- The PropTypes should always be defined. This will help is track all props in the app and it will also be useful for any developer working on the same project.

- Try to write application logic in render method. Do any kind of processing from state or props in render method only.

- Follow a single responsibility principle.ie use one component to do one task/functionality only.

- Don’t unnecessarily use context or have your application tied to it

- If performance is a concern—avoid recreating functions or objects in your render
- Don’t use state when you can use props or local instance variables
- Prior react versions used mixins for handling reusable functionalities.As they are deprecated now, alternate solution has been provided, nothing but HOC(Higher Order Components)
- React Dev Tools is an excellent way to explore our React components and helps diagnose any issues in your app.
- Use Inline Conditional Statements
- Stateless functional components can be used when there is no need for **state**, **refs**, or **lifecycle methods**. Use it only to return jsx.
- Use PureComponent rather than a Component to prevent things from having an unnecessary re-render.

Demo

Demos of

React-lifecycle-old
react-lifecycle-2019
react-lifecycle-2019_2

Review:

What are two ways the get values from <FORM> element?

1. Contolled components
2. Nested Components
3. input ref’s
4. routing

The _____ operator is called as spread operator.

1. “…”
2. “..”
3. []
4. !.!