React - advanced

adapt

React Children

React allows nesting Components inside each other.

```
const Main = () => (
  <Wrapper>
    <SubComponent/>
  </Wrapper>
);

const Wrapper = (children) => (
  <div>
    {children}
  </div>
);
```

adapt

React.Children

React.Children provides utilities for dealing with the this.props.children opaque data structure.

- React.Children.map
- React.Children.forEach
- React.Children.count
- React.Children.only
- React.Children.only
- React.Children.toArray

React.Fragment

The React.Fragment component lets you return multiple elements in a render() method without creating an additional DOM element.

```
const Example = (
  <React.Fragment>
    Some text.
    <h2>A heading</h2>
  </React.Fragment>
);
```

adapt

Controlled components

Uncontrolled components, where form data is handled by the DOM itself.

Controlled components, where form data is handled by component itself.

```
const Example = (
  <form onSubmit={this.handleSubmit}>
    <label>
      Name:
      <input type="text" value={this.state.value} onChange={this.handleChange} />
    </label>
    <input type="submit" value="Submit" />
  </form>
);
```

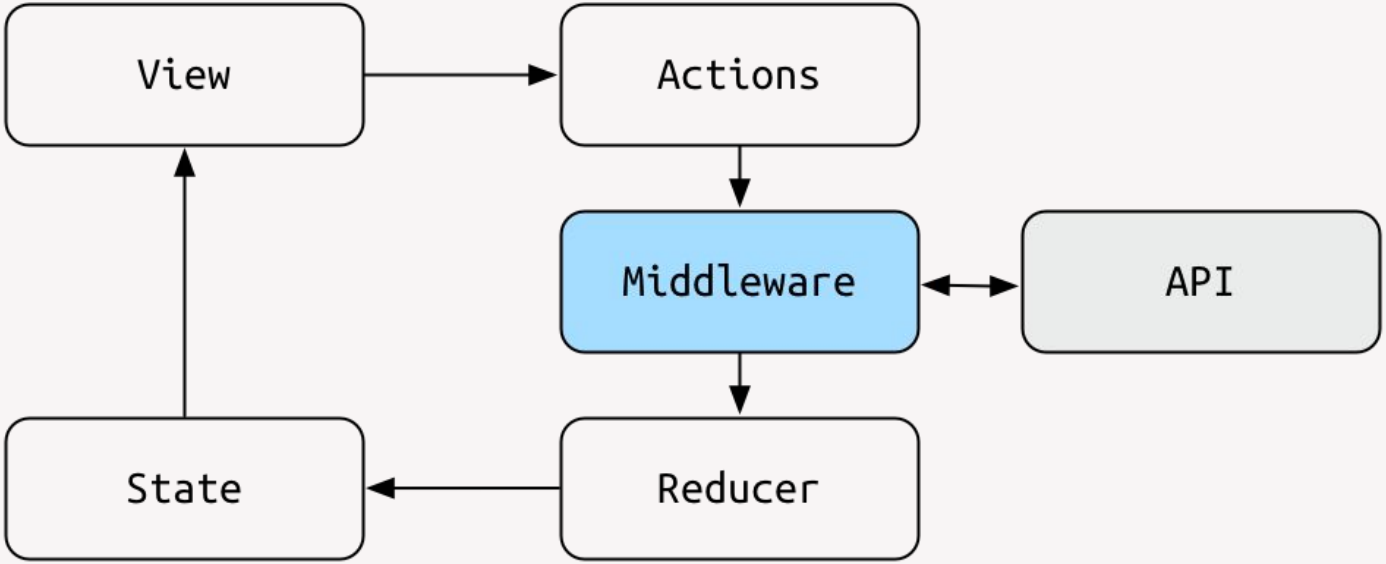
adapt

Redux Middleware

Middleware is some code you can put between the framework receiving a request, and the framework generating a response.

Redux middleware provides a third-party extension point between dispatching an action, and the moment it reaches the reducer.

Redux Middleware



adapt

Example

Refs and the DOM

Refs provide a way to access DOM nodes or React elements created in the render method.

In the typical React dataflow, props are the only way that parent components interact with their children. To modify a child, you re-render it with new props. However, there are a few cases where you need to imperatively modify a child outside of the typical dataflow. The child to be modified could be an instance of a React component, or it could be a DOM element. For both of these cases, React provides an escape hatch.

Refs and the DOM

When to Use Refs:

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

Don’t Overuse Refs

Refs and the DOM

There is two ways to set ref:

1. Creating Refs using React.createRef()
2. Creating Refs using callback functions.

Example

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, per se. They are a pattern that emerges from React's compositional nature.

Concretely, a higher-order component is a function that takes a component and returns a new component.

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

Higher-Order Components

```
import React from 'react';

const withSecretToLife = (WrappedComponent) => {
  class HOC extends React.Component {
    render() {
      return (
        <WrappedComponent
          {...this.props}
          secretToLife={42}
        />
      );
    }
  }

  return HOC;
};

export default withSecretToLife;
```

adapt

Higher-Order Components

Recommendations:

- Don’t Mutate the Original Component
- Pass Unrelated Props Through to the Wrapped Component
- Maximizing Composability
- Wrap the Display Name for Easy Debugging
- Don’t Use HOCs Inside the render Method
- Static Methods Must Be Copied Over
- Refs Aren’t Passed Through

Example

Recompose

Recompose is a React utility belt for function components and higher-order components. Think of it like lodash for React.

Two main utilities that you should know:

- compose
- pure

Must read:

https://github.com/vasanthk/react-bits

Questions?