

Higher Order Component

Friday, March 25, 2022 2:56 PM

- Consider HOC as wrapper (or parent or container) for your component
- It wraps your component
- Once your Component becomes a Child it can
 - Receive properties from the wrapper (HOC)
 - Hoc can add additional JSX before or after your Child component

Higher Order Component (HOC)

- ▶ Consider We Create our own component
- ▶ HOC can be considered as a parent component.
- ▶ As a parent it
 - ▶ can add additional JSX
 - ▶ Can pass additional properties

```
const Hoc = (props) =>{
  const something=...
  return ( <div>
    <h2>{props.screenName}</h2>
    <UserComponent newProp={something} />
  </div>;
}
```

What makes HOC different from Parent?

- HOC + MyComponent = MyEnhancedComponent
- User will be using the combination as an enhanced alternative my basic unwrapped component.
- They will not think in terms of HOC + MyComponent
- They will think of them as a combined single UNIT.
- HOC enhances your component. The two becomes inseparable single entity.

Analogy

- You can think of a Mobile Cover as an HOC for the Mobile
- When you apply the cover, you get a protected mobile phone.
- You use the combination as a single unit and don't really view them differently in your daily uses.
 - If you ask someone to get your mobile phone to you, what would they bring?
 - Only mobile?

HOC Composer

- We don't combine our component and HOC together manually
- We use a composer function which will generally have name **withXyz**.

```
var mobile = new Mobile(); //bare naked mobile
var coveredMobile = withCover(mobile); //covered mobile
var temperProofCoveredMobile = withTemperproofGlass( coveredMobile) ;
```

- **"with"** tells you get your component with extra feature.

How do you control volume of a covered mobile

- Cover should ensure that any external interaction (props/events) initiated by the user on cover should reach actual mobile
 - You press volume button on the cover (you pass props to HOC)
 - The cover passes the same press to the original mobile

Cover HOC With Mobile

```
let Mobile = (props) =>{
  return <span>Mobile Phone {props.brand}</span>
}
```

```
const withCover = (Component) => {
  const Cover = () => {
    <div>
      <span>[</span>
        <Component />
      <span>]</span>
    </div>
  };
  return Cover;
}
```

What does this code do ?

Mobile=withCover(Mobile);

<Mobile brand = "samsung" />

```
const withCover = (Component) => {
  const Cover = (props) => {
    <div>
      <span>[</span>
        <Component brand={props.brand} />
      <span>]</span>
    </div>
  };
  return Cover;
};

export default withCover;
```

- Q. What Component is really represented by word Mobile
- Mobile is actually Cover
 - Component is the actual Mobile
- Q. Who received the brand props?
- Cover
- Q. What did the Cover do with the brand prop?
- Nothing
- Q. How can we pass 'props' to actual Mobile Component (Not Cover)
- We CAN'T. We can pass props to any nested child
 - WE can pass props to Cover
 - Cover should pass it to Component

What if there are other properties

- What about mobile model or camera?
- How can HOC pass parameters that are specific to component and it doesn't know about

We can pass all parameter using object DE structuring

```
const withCover = (Component) => {
  const Cover = (props) => {
    <div>
      <span>[</span>
        <Component {...props} />
      <span>]</span>
    </div>
  };
  return Cover;
};

export default withCover;
```

A Minimal HOC design (that does NOTHING)

```
const withDoNothing = (Component) => {
  const DoNothing = (props) => {
    return <Component {...props} />
  };
}
```

```
}  
  return DoNothing;  
}
```

withDate Hoc

```
const withDate = (Component)=>{  
  return (props)=>{  
    var date=new Date();  
    return <Component {...props} date={date} />;  
  }  
}  
  
export default withDate;
```

- All property passed by the parent compoent

- Additional info injected by the HOC