Hasta ahora hemos trabajado con dos tipos de componentes: los de clase o *stateful* y los de función o *stateless*. En React existen dos tipos más, así como una división lógica que se suele hacer en las aplicaciones con el objetivo de simplificar el desarrollo.

Tipos en React

Stateful

Las principales características de este tipo de componentes son:

- Se definen como clases y pueden definir y actualizar su estado
- Cada actualización de los props o el state provoca una llamada al método `render`
- Podemos sobrescribir los métodos de su ciclo de vida.

Stateless

Podemos considerar estos componentes como una versión simplificada de los stateful.

- Se definen como funciones
- Cada cambio en los props provoca su ejecución
- No definen estado
- No permiten sobrescribir los métodos de su ciclo de vida

Internamente, React transforma estos componentes a stateful, para tratar a todos los elementos por igual. En el futuro, estos componentes tendrán ciertas optimizaciones a la hora de trabajar con ellos.

PureComponent

Este tipo de componentes es similar a los stateful en cuanto a su definición. También se definen como clases, solo que esta extiende de `React.PureComponent` . La mayor diferencia es que este componente tampoco define estado y sí que está optimizado en React.

El método `shouldComponenteUpdate` de este tipo de componentes sólo retorna `true` cuando recibe nuevos props y los valores son distintos a los anteriores.

- Se definen como clases
- No tienen estado
- Solo se llama al método render cuando hay nuevos props y estos son distintos a los anteriores
- Podemos sobrescribir los métodos de su ciclo de vida

```
// Este componente solo se actualiza cuando cambian los valores de los props
class PureTitle extends React.PureComponent {
  render() {
    return <h1>{ this.props.title }</h1>
  }
}
```

En este ejemplo de Codepen podéis ver en la consola cuando se actualizan los componentes.

High Order Components (HOC)

Este tipo de componentes es en realidad una función que recibe un componente como parámetro y genera un nuevo componente. Si pensamos simplemente en funciones, podemos ver los HOC como una función que modifica un parámetro antes de ser pasado a otra.

```
// Tenemos una función simple
const hello = (word) => `Hello ${word}!`;
// Creamos nuestro HOC
const uppercaseHOC = (word) => word.toUpperCase();

hello('World') // Hello World!
hello(uppercaseHOC('World')) // Hello WORLD!
```

Si analizamos este ejemplo, **`uppercaseHOC` es una función que toma un parámetro y retorna dicho elemento transformado**. `hello` puede ser utilizado con o sin `uppercaseHOC` . Otra opción sería pasar un parámetro opcional a `hello` que indicase si queremos nuestra palabra en mayúsculas. No obstante, `uppercaseHOC` es reutilizable ya que podemos modificar los parámetros de otras funciones.

Si extrapolamos este concepto a componentes, **un HOC es una función que recibe un componente como parámetro y genera uno nuevo**. Este tipo de componentes es muy útil para extraer un comportamiento común y aplicarlo a distintos componentes. Los nuevos componentes son stateful. Tienen sus props, su ciclo de vida y su estado

propio. Un HOC toma unos props y genera un nuevo objeto con los valores modificados para el componente principal.

Como ejemplo, vamos a crear un HOC que recorte descripciones a un número concreto de caracteres.

```
// Nuestro componente es una función. WrappedComponent es el componente
// principal
const TruncateHOC = WrappedComponent =>
  class extends React.Component {
    // El prop text es requerido
    static propTypes = {
      text: React.PropTypes.string.isRequired
    };

    // Recorta el text si el tamaño supera maxLength
    truncateText(text) {
      const maxLength = 25;

      if (text.length > maxLength) {
        return `${ text.substring(0, maxLength - 3) }...`;
      } else {
        return text;
      }
    }

    // Renderizamos el componente
    render() {
      // Creamos una copia de los props actuales
      let props = Object.assign({}, this.props);
      // Modificamos el texto
      props.text = this.truncateText(props.text);

      // Renderizamos el componente principal
      return <WrappedComponent { ...props } />;
    }
  };
```

Una vez tenemos definido este HOC, ya podemos aplicarlo a todos los componentes que necesitemos.

```
// Componente de descripción
let Description = props => <p>{ props.text }</p>

// Otro componente para representar artículos
class Article extends React.Component {
  render() {
    return <article>{ this.props.text }</article>
  }
}

// Aplicamos el HOC!
Description = TruncateHOC(Description);
Article = TruncateHOC(Article);
```

Código en Codepen.