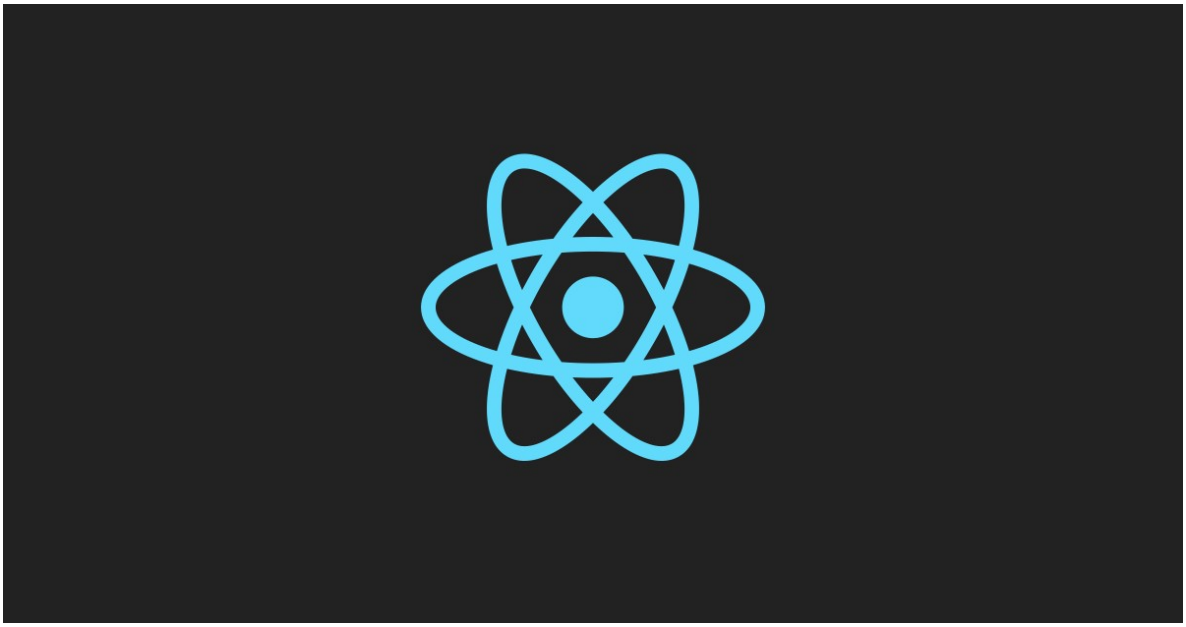
React JS en español
Primeros pasos



Autor: Wilson Flores Turriate

URL del post: https://frontendlabs.io/3158--react-js-espanol-tutorial-basico-primeros-pasos-ejemplos

En este post se le hablará acerca de los primeros pasos en React js, los puntos a exponer son los siguientes.

- ¿Que es React js?
- Componentes en React js
- Sintaxis JSX
- Virtual Dom
- Props y State
- Ciclo de vida de los componentes

¿QUE ES REACT JS?

React js es una librería Javascript creada ya hace un tiempo por Facebook, para construir interfaces de usuario, que te permitan crear aplicaciones SPA (single page application) más eficientes y funciona tanto en el lado cliente como en el servidor, haciendo posible la creación de aplicaciones isomórficas.

Al momento de empezar un nuevo proyecto en javascript, siempre tratamos de buscar las mejores herramientas, patrones de diseño, frameworks o librerías para desarrollarlo, entre los más populares patrones de diseño para crear aplicaciones web podemos ver que se encuentra el MVC(Model-View-Controller), menciono el patrón MVC porque React js es muy usado para hacer la V(View) en MVC. React js se encarga principalmente del renderizado de las vistas, también fomenta la creación de componentes de interfaz de usuario reutilizables que presentan cambios en el tiempo.

Ahora nos toca preparar nuestro terreno para poder desarrollar con React js, tal cual se recomienda en su guía oficial, el uso de estándar CommonJS, para ello puedes usar Browserify o Webpack.

En primer lugar vamos a definir nuestro árbol de carpetas necesarias para empezar.

```
|-- 1-hello-world
        |-- app
            |-- components
                |-- main.jsx
        |-- public
            |-- index.html
        |-- package.json
        |-- webpack.config.js
```

Para este ejemplo se usarán los siguientes paquetes npm.

REACT JS

Es el paquete que nos trae toda la API de React js.

REACT-DOM

Nos permite interactuar con el DOM.

BABEL-CORE

Es el núcleo compilador de babel.

BABEL-LOADER

Este paquete nos permitirá trabajar con webpack.

BABEL-PRESET-ES2015

Este paquete nos permite codear con ecmascript 6.

BABEL-PRESET-REACT

Este paquete nos permite codear jsx .

WEBPACK

Es nuestro empaquetador de módulos.

Los conceptos de estos paquetes se quedan muy cortos, es por eso que se explicarán a detalle sobre estos paquetes en nuestros próximos posts.

Luego ejecutar el siguiente comando para poder instalarlos.

```
npm install react react-dom babel-core babel-loader babel-preset-es2015 babel-preset-react webpack --save-dev
```

Un html simple

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
```

```
    <div id="container"></div>
    <script type="text/javascript" src="build.js"></script>
  </body>
</html>
```

Dentro de nuestro package.json incluir el script siguiente.

```
"scripts": {
    "build": "webpack -w",
  },
```

Nuestra configuración webpack.config.js

```
module.exports = {
  entry: './app/components/main.jsx',
  output: {
    path: './public/',
    filename: "build.js",
  },
  module: {
    loaders: [
      {
        exclude: /(node_modules|bower_components)/,
        loader: 'babel',
        query: {
          presets: ['es2015', 'react']
        }
      }
    ]
  }
};
```

Con esto tenemos todo listo para nuestros siguientes ejemplos, sólo deben ejecutar npm run build para transpilarlo.

COMPONENTES EN REACT JS

Un componente viene a ser cada uno de los elementos de nuestra aplicación, como por ejemplo: el header, footer, botones, entre otros elementos de nuestra UI.

CREANDO NUESTRO PRIMER COMPONENTE EN REACT JS

Un componente se crea en ES5 con **React.createClass({…})**, usted debe proporcionar un objeto que contiene un método render, también pueden contener opcionalmente otros métodos del ciclo de vida, que se explicará más adelante. El método render **render:function(){…}** es el método que React js llamará para poder pintar el componente, este es obligatorio.

COMPONENTE HELLO WORLD CON REACT JS

(ES5)

```
var React = require('react');
var ReactDOM = require('react-dom');

var HelloWorld = React.createClass({
  render: function(){
    return(<div>Hello World!</div>);
  }
});

ReactDOM.render(
  <HelloWorld />,
  document.getElementById('container')
);
```

También podemos crear un componente con la sintaxis ES6.

ES6 CLASSES

```
import React from 'react';
import ReactDOM from 'react-dom';

class HelloWorld extends React.Component{
  render(){
    return <div>Hello world!</div>
  }
}

ReactDOM.render(
  <HelloWorld />,
  document.getElementById('container')
);
```

SINTAXIS JSX

Es una extensión de la sintaxis XML, claro está que no es la intención de ser implementado por motores o navegadores.
Está destinado a ser utilizado por varios preprocesadores(transpiler) para transformarlo en un estándar ECMAScript.

No necesariamente tiene que usar JSX con React js,si usted desea puede usar solo javascript, claro está que se recomienda utilizar JSX por que es más legible a la hora de desarrollar.

Para transpilar JSX puedes usar babel.

Sin JSX

```
var HelloWorld = React.createClass({
  render: function(){
    return (
      React.createElement('div', { className: 'hello' },
'Hello World!')
    );
  }
});
```

Con JSX

```
var HelloWorld = React.createClass({
  render: function(){
    return (
      <div className="hello">Hello World!</div>
    );
  }
});
```

VIRTUAL DOM

A veces modificar el DOM termina siendo un proceso muy costoso, por suerte React js hace todo este trabajo por nosotros, pero ¿Comó lo hace?, Acá es donde entra el Virtual DOM, pero ¿Qué es?, en React js los componentes no generan HTML directamente, lo que generan es código Javascript, una descripción virtual del DOM, al realizar el renderizado, React js utiliza un algoritmo de diff para comparar la salida de los componentes con el virtual DOM actual para ver qué debe cambiar en el DOM real, esto lo hace muy eficientemente, ya que sólo se realizarán las operaciones que sean estrictamente necesarias.

PROPS

Al igual que hemos visto en los elementos html, donde podías agregarle atributos en su etiqueta de entrada, también hay formas para personalizar los componentes de react js, digamos que las propiedades son para los componentes de React js como los atributos son a los elementos html.

(ES5)

```
var React = require('react');
var ReactDOM = require('react-dom');

var MyComponent = React.createClass({
  render : function (){
    return (<div>MyComponent {this.props.name}</div>)
  }
})

ReactDOM.render(
  <MyComponent name="Jupiter" />,
  document.getElementById('container')
);
```

(ES6)

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyComponent extends React.Component{
  render(){
    return <div>MyComponent {this.props.name}</div>
  }
}

ReactDOM.render(
  <MyComponent name="Jupiter" />,
  document.getElementById('container')
);
```

Ahora veamos cómo podemos establecer propiedades por defecto.

(ES5)

```
var React = require('react');
var ReactDOM = require('react-dom');

var MyComponent = React.createClass({
  getDefaultProps : function(){
    return{
      name:'Marte'
    }
  },
  render : function (){
    return (<div>MyComponent {this.props.name}</div>)
  }
})

ReactDOM.render(
  <MyComponent />,
  document.getElementById('container')
);
```

(ES6)

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyComponent extends React.Component{
  render(){
    return (<div>MyComponent {this.props.name}</div>)
  }
}

MyComponent.defaultProps = {
  name:'Saturno'
};

ReactDOM.render(
  <MyComponent />,
  document.getElementById('container')
);
```

Nuestro **getDefaultProps** se ha transformado en **(ES6)** para pasar a ser una propiedad de objeto.

STATE

A diferencia de las propiedades que se definen tanto por JSX o Javascript, los estados sólo se definen en el interior del componente. Para poder utilizar los estados se debe declarar un conjunto predeterminado de valores, para ello se utiliza el método llamado **getInitialState**

(ES5)

```
var React = require('react');
import ReactDOM = require('react-dom');

var MyComponent = React.createClass({
  getInitialState: function() {
    return {
      name:'Jupiter'
    };
  },

  render: function() {
    return (<div>My Component {this.state.name}</div>);
  }
});

ReactDOM.render(
  <MyComponent />,
  document.getElementById('container')
);
```

(ES6)

En es6 todos los valores predeterminados declarados en getInitialState serán declarados en nuestro constructor como una simple propiedad de inicialización.

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyComponent extends React.Component{
  constructor(props){
    super(props);
    this.state = {name:'Ganimedes'};
  }
  render(){
    return <div>My Component {this.state.name}</div>;
  }
```

```
};

ReactDOM.render(
  <MyComponent />,
  document.getElementById('container')
);
```

Que pasa si queremos setear nuestro state en un futuro con alguna acción, es donde el método this.setState entra en escena, este es el método principal que se utiliza para hacer cambios en la interfaz de usuario.

(ES5)

```
var React = require('react');
var ReactDOM = require('react-dom');

var MyComponent = React.createClass({
  getInitialState:function(){
    return{
      message:'Jupiter'
    }
  },
  update:function(){
    this.setState({message:'Saturno'});
  },
  render:function(){
    return(<div>
      <span>My Component {this.state.message}<br></br></span>
      <button onClick={this.update}>click me</button>
    </div>)
  }

})

ReactDOM.render(
  <MyComponent />,
  document.getElementById('container')
);
```

(ES6)

```
import React from 'react';
import ReactDOM from 'react-dom';
```

```
class MyComponent extends React.Component{
  constructor(props){
    super(props);
    this.state = {message:'Jupiter'};
    this.update = this.update.bind(this);
  }

  update(){
    this.setState({message:'Saturno'})
  }

  render(){
    return <div>
       <span>My Component
{this.state.message}<br></br></span>
      <button onClick={this.update}>click me</button>
    </div>
  }
}

MyComponent.defaultProps = {
  val:0
};

ReactDOM.render(
  <MyComponent />,
  document.getElementById('container')
);
```

CICLO DE VIDA DE LOS COMPONENTES

Se definen 3 etapas de un componente.

MOUNTING

Cuando el componente se está montando en el Dom.

UPDATING

Cuando se están modificando los props y states.

UNMOUNTING

Cuando el componente se está quitando del Dom.

Existen varios métodos que nos provee React js que nos ayudará a la hora de crear nuestros componentes para distintos escenarios.

MOUNTING: COMPONENTWILLMOUNT

Se invoca una vez tanto en el cliente y el servidor. Se ejecuta antes de que el componente sea montado en el DOM.

(ES5)

```
import React from 'react';
import ReactDOM from 'react-dom';

var MyComponent = React.createClass({
  componentWillMount:function(){
    console.log('Before mount...');
  },
  render:function(){
    return(<div>MyComponent</div>)
  }
});

ReactDOM.render(
  <MyComponent />,
  document.getElementById('container')
);
```

(ES6)

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyComponent extends React.Component {
  constructor(props){
    super(props);
  }
  componentWillMount(){
    console.log('Before mount...');
  }
  render(){
    return <div>My Component</div>
  }
}
```

```
ReactDOM.render(
  <MyComponent />,
  document.getElementById('container')
);
```

El método **constructor** se ejecuta antes **componentWillMount**.

MOUNTING: COMPONENTDIDMOUNT

Se ejecuta inmediatamente después que sea montado en el DOM. Si deseas hacer peticiones AJAX o integrar con librerías de terceros como jquery u otros, este método es el ideal.

(ES5)

```
var React = require('react');
var ReactDOM = require('react-dom');

var MyComponent = React.createClass({
  componentDidMount:function(){
    console.log('Mount');
  },

  render:function(){
    return (<div>MyComponent</div>)
  }
});

ReactDOM.render(
  <MyComponent />,
  document.getElementById('container')
);
```

(ES6)

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyComponent extends React.Component {
  constructor(props){
    super(props);
    console.log('Before mount');
  }
```

```
  componentDidMount(){
    console.log('Mount');
  }

  render(){
    return <div>MyComponent</div>
  }
}

ReactDOM.render(
  <MyComponent />,
  document.getElementById('container')
);
```

UPDATING: COMPONENTWILLRECEIVEPROPS

Este método es invocado cuando un componente está recibiendo nuevas props. Este método no es invocado en el primer render.

(ES5)

```
var = React = require('react');
var = ReactDOM = require('react-dom');

var MyComponent = React.createClass({
  componentWillReceiveProps:function(nextProps) {
    console.log('nextProps ',nextProps.name);
    console.log('props ',this.props.name);
  },
  render:function(){
    return <div>Bar {this.props.name}!</div>;
  }
});

ReactDOM.render(<MyComponent name="jupiter" />,
document.getElementById('container'));

ReactDOM.render(<MyComponent name="saturno" />,
document.getElementById('container'));

ReactDOM.render(<MyComponent name="ganimedes" />,
document.getElementById('container'));
```

(ES6)

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyComponent extends React.Component {
  componentWillReceiveProps(nextProps) {
    console.log('nextProps ',nextProps.name);
    console.log('props ',this.props.name);
  }
  render() {
    return <div>Bar {this.props.name}!</div>;
  }
}

ReactDOM.render(<MyComponent name="jupiter" />,
document.getElementById('container'));

ReactDOM.render(<MyComponent name="saturno" />,
document.getElementById('container'));

ReactDOM.render(<MyComponent name="ganimedes" />,
document.getElementById('container'));
```

Como se darán cuenta se invoca 2 veces el método **componentWillReceiveProps**, por qué estamos actualizando 2 veces el props.

UPDATING: SHOULDCOMPONENTUPDATE

Se invoca antes del render, cuando se estén recibiendo nuevos props o states. Siempre devolverá true, si hacemos que retorne false evitaremos el render hasta un nuevo cambio en las props o state.

(ES5)

```
var React = require('react');
var ReactDOM = require('react-dom');

var MyComponent = React.createClass({
  getDefaultProps:function(){
    return{
      val:0
    }
  },

  shouldComponentUpdate:function(nextProps,nextState){
```

```
    console.log(nextProps);
    return nextProps.val % 2 == 0;
  },

  componentWillUpdate:function(){
    console.log('Update MyComponent...');
  },

  update:function(){
    ReactDOM.render(
      <MyComponent val={this.props.val + 1}/>,
      document.getElementById('container')
    );
  },

  render:function(){
    return (<div>
      <span>My Component contador:</span>
      <button onClick={this.update}>{this.props.val}</button>
    </div>)
  }
});

ReactDOM.render(
  <MyComponent />,
  document.getElementById('container')
);
```

(ES6)

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyComponent extends React.Component {
  constructor(props){
    super(props);
    this.update = this.update.bind(this);
  }

  shouldComponentUpdate(nextProps,nextState){
    console.log(nextProps);
    return nextProps.val % 2 == 0;
  }

  componentWillUpdate(){
```

```
    console.log('Update MyComponent...');
  }

  update(){
    ReactDOM.render(
      <MyComponent val={this.props.val + 1}/>,
      document.getElementById('container')
    );
  }
  render(){
    return <div>
      <span>My Component contador:</span>
      <button onClick={this.update}>{this.props.val}</button>
    </div>
  }
}

MyComponent.defaultProps = {
  val:0
};

ReactDOM.render(
  <MyComponent />,
  document.getElementById('container')
);
```

En este ejemplo solo nos renderiza el componente cuando **this.props.val** dividido entre dos, el cociente sea 0, para este ejemplo hacemos uso del evento **onClick** de React js.

UPDATING: COMPONENTWILLUPDATE

Se invocará antes del render, justo antes que tu componente se haya actualizado (recibiendo nuevas props o state).
Excelente para procesos que necesiten hacer antes de hacer la actualización.

(ES5)

```
var MyComponent = React.createClass({
  componentWillUpdate:function(){
    console.log('Update MyComponent');
  },
  render:function(){
    return <div>my Component {this.props.name}</div>
  }
});
```

```
ReactDOM.render(<MyComponent
name="jupiter"/>,document.getElementById('container'));
ReactDOM.render(<MyComponent
name="neptuno"/>,document.getElementById('container'));
```

(ES6)

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyComponent extends React.Component{
  componentWillUpdate(){
    console.log('Update MyComponent');
  }
  render(){
    return <div>my Component {this.props.name}</div>
  }
}

ReactDOM.render(<MyComponent
name="jupiter"/>,document.getElementById('container'));
ReactDOM.render(<MyComponent
name="neptuno"/>,document.getElementById('container'));
```

UPDATING: COMPONENTDIDUPDATE

Se invoca inmediatamente después del render, justo cuando tu componente ha cambiado.

(ES5)

```
var React = require('react');
var ReactDOM = require('react-dom');

var MyComponent = React.createClass({
  componentDidUpdate:function(prevProps,prevState){
    console.log('prevPros or prevState');
  },
  render:function(){
    return <div>my Component {this.props.name}</div>
  }
});
```

```
ReactDOM.render(<MyComponent
name="jupiter"/>,document.getElementById('container'));
ReactDOM.render(<MyComponent
name="neptuno"/>,document.getElementById('container'));
```

(ES6)

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyComponent extends React.Component{
  componentDidUpdate(prevProps,prevState){
    console.log('prevPros or prevState');
  }
  render(){
    return <div>my Component {this.props.name}</div>
  }
}

ReactDOM.render(<MyComponent
name="jupiter"/>,document.getElementById('container'));
ReactDOM.render(<MyComponent
name="neptuno"/>,document.getElementById('container'));
```

UNMOUNTING: COMPONENTWILLUNMOUNT

Se invoca inmediatamente antes de que un componente sea desmontado del DOM. Aquí puedes realizar la limpieza de cualquier referencia de memoria que nos hayan quedado.

(ES5)

```
var React = require('react');
var ReactDOM = require('react-dom');

var MyComponent = React.createClass({
  componentWillMount:function(){
    console.log('Mount');
  },
  componentWillUnmount:function(){
    console.log('Unmount');
  },
  render:function(){
    console.log('Rendering');
```

```
    return <div>I am component</div>
  }

});

var MyApp = React.createClass({
  mount:function(){
    ReactDOM.render(<MyComponent
/>,document.getElementById('component'));
  },
  unmount:function(){

ReactDOM.unmountComponentAtNode(document.getElementById('comp
onent'));
  }
  render:function(){
    return(
      <div>
        <button onClick={this.mount.bind(this)}>Mount
component</button>
        <button onClick={this.unmount.bind(this)}>Unmount
component</button>
        <div id="component"></div>
      </div>
    )
  }
});

ReactDOM.render(<MyApp />,
document.getElementById('container'));
```

(ES6)

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyComponent extends React.Component {
  constructor(props){
    super(props);
  }

  componentWillMount(){
    console.log('Mount');
  }
  componentWillUnmount(){
```

```
    console.log('Unmount');
  }
  render(){
    console.log('Rendering');
    return <div>I am component</div>
  }
}

class MyApp extends React.Component{
  constructor(pros){
    super(pros);
  }

  mount(){
    ReactDOM.render(<MyComponent
/>,document.getElementById('component'));
  }

  unmount(){

ReactDOM.unmountComponentAtNode(document.getElementById('comp
onent'));
  }

  render(){
    return(
      <div>
        <button onClick={this.mount.bind(this)}>Mount
component</button>
        <button onClick={this.unmount.bind(this)}>Unmount
component</button>
        <div id="component"></div>
      </div>
    )
  }
}

ReactDOM.render(<MyApp />,
document.getElementById('container'));
```

En este post se ha tratado de hablar de todo un poco para que puedas empezar con React js, los ejemplos y el árbol de directorio empleado para el desarrollo de los ejemplos, sólo fueron empleados para ello, usted puede emplear el que sea de su agrado. Como ven aparte de los puntos expuestos se ha hablado de otros temas de React js, como eventos y un método de desmontaje de componentes, como también sobre la sintaxis ES6 para algunos ejemplos, diferencias que no an sido expuestas con respecto a ES5, temas que abarcaremos en otros posts, sin más muchas gracias por haber leído este post.

Si alguno de los ejemplos no les funcionó pueden revisar el repositorio del post donde podrás encontrar todos los ejemplos mostrados. Repositorio en github.