BackboneJS

Sesión 7 - Interfaces web con React (II)

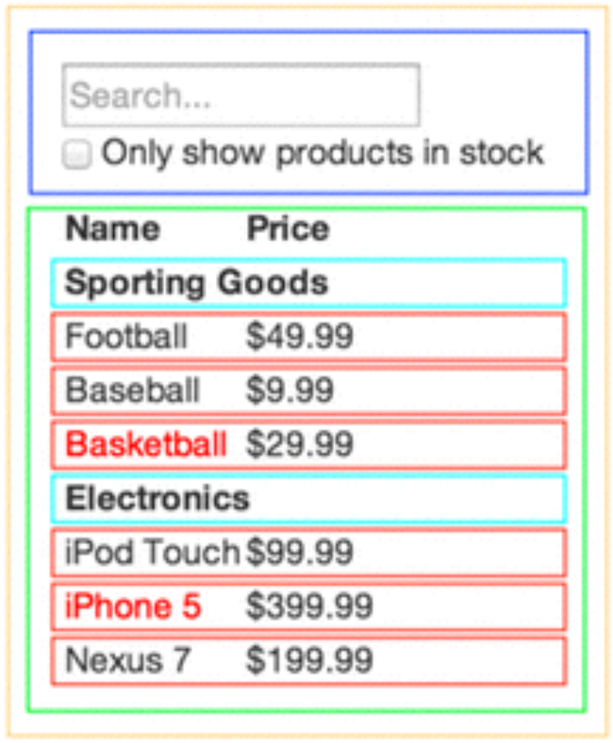
© 2015-2016 Depto. Ciencia de la Computación e IA

Índice

© 2015-2016 Depto. Ciencia de la Computación e IA

Jerarquías de componentes

- Normalmente tendremos componentes que contienen otros componentes



- `FilterableProductTable`
  - `SearchBar`
  - `ProductTable`
    - `ProductCategoryRow`
    - `ProductRow`

(del tutorial "Thinking in React", de Pete Hunt)

© 2015-2016 Depto. Ciencia de la Computación e IA

Lista de la compra con jerarquía de componentes (sin estado)

```
var Item = React.createClass({
  render: function() {
    return <li> {this.props.nombre} ({this.props.cantidad}) </li>
  }
});

var ListaCompra = React.createClass({
  render: function() {
    var items = this.props.items.map(function(item, indice) {
      return <Item key={indice} nombre={item.nombre} cantidad={item.cantidad}/>
    })
    return <ul>{items}</ul>;
  }
});
var lista = [{nombre:'huevos', cantidad:12}, {nombre:'pan', cantidad:1}];
var instancia = <ListaCompra items={lista}/>;
ReactDOM.render(instancia, document.getElementById('miApp'));
```

Índice

© 2015-2016 Depto. Ciencia de la Computación e IA

¿Dónde colocamos el estado?

- Yo probablemente haría que cada `Item` mantuviera su propio estado, PERO…
- **Buena práctica** aceptada en React: reducir el número de componentes con estado al mínimo posible
- **Razón**: los componentes sin estado son más fáciles de *testear*. "Solo" hay que comprobar que dado un cierto valor de props generan el HTML correcto

© 2015-2016 Depto. Ciencia de la Computación e IA

Reglas para colocar el estado (de *Thinking in React)*

- Identificar todos los componentes que renderizan algo basándose en esa variable
- Encontrar un componente dueño común a todos ellos (es decir, un único componente por encima en la jerarquía de todos los que necesitan ese estado).
- El estado debería residir en ese dueño común o en otro componente todavía más alto en la jerarquía.
- Si no podemos encontrar un componente en el que pueda residir el estado, crear uno nuevo simplemente para almacenarlo, y añadirlo en la jerarquía en algún lugar por encima de ese dueño común.

© 2015-2016 Depto. Ciencia de la Computación e IA

En resumen

Colocar el estado lo más alto posible en la jerarquía. En un caso extremo crear un "componente raíz" que no tenga interfaz y solo sirva para almacenar estado y lógica

© 2015-2016 Depto. Ciencia de la Computación e IA

Lista de la compra con jerarquía y estado

http://jsbin.com/gojezevuge/edit?js,output

```
var Item = React.createClass({
   render: function() {
     if (this.props.comprado) {
       return <li className="tachado">{this.props.nombre} ({this.props.cantidad})</li>
     }
     else {
       return <li>{this.props.nombre} ({this.props.cantidad})</li>
     }
   }
});
var ListaCompra = React.createClass({
   getInitialState: function() {
     return {comprados: new Array(this.props.items.length)};
   },
   render: function() {
      var items = this.props.items.map(function(item, indice) {
        return (<Item key={indice}
                nombre={item.nombre} cantidad={item.cantidad} comprado={this.state.comprados[indice]}/>)
      }.bind(this));
      return <ul>{items}</ul>;
   }
  });
```

Índice

© 2015-2016 Depto. Ciencia de la Computación e IA

Comunicación abajo-arriba

- Normalmente **el estado estará arriba** en la jerarquía
- Pero los eventos normalmente afectan a los componentes de "abajo"
- Los componentes **hijos deben "avisar" al padre** del cambio de estado. Lo más habitual en React es llamar a un *callback* del padre, que es el que cambia el estado

http://jsbin.com/juwowixafo/1/edit?js,output

© 2015-2016 Depto. Ciencia de la Computación e IA

Comunicación padre-hijo

- El padre le pasa al hijo una función que actualiza el estado, como una “prop” más

```
render: function() {
    var items = this.props.items.map(function(item, indice) {
      return <Item id={indice} key={indice}
        nombre={item.nombre}
        cantidad={item.cantidad}
        comprado={this.state.comprados[indice]}
        handleClick={this.toggleState}/>
    }.bind(this));
    return <ul>{items}</ul>;
}
```


© 2015-2016 Depto. Ciencia de la Computación e IA

Comunicación padre-hijo (II)

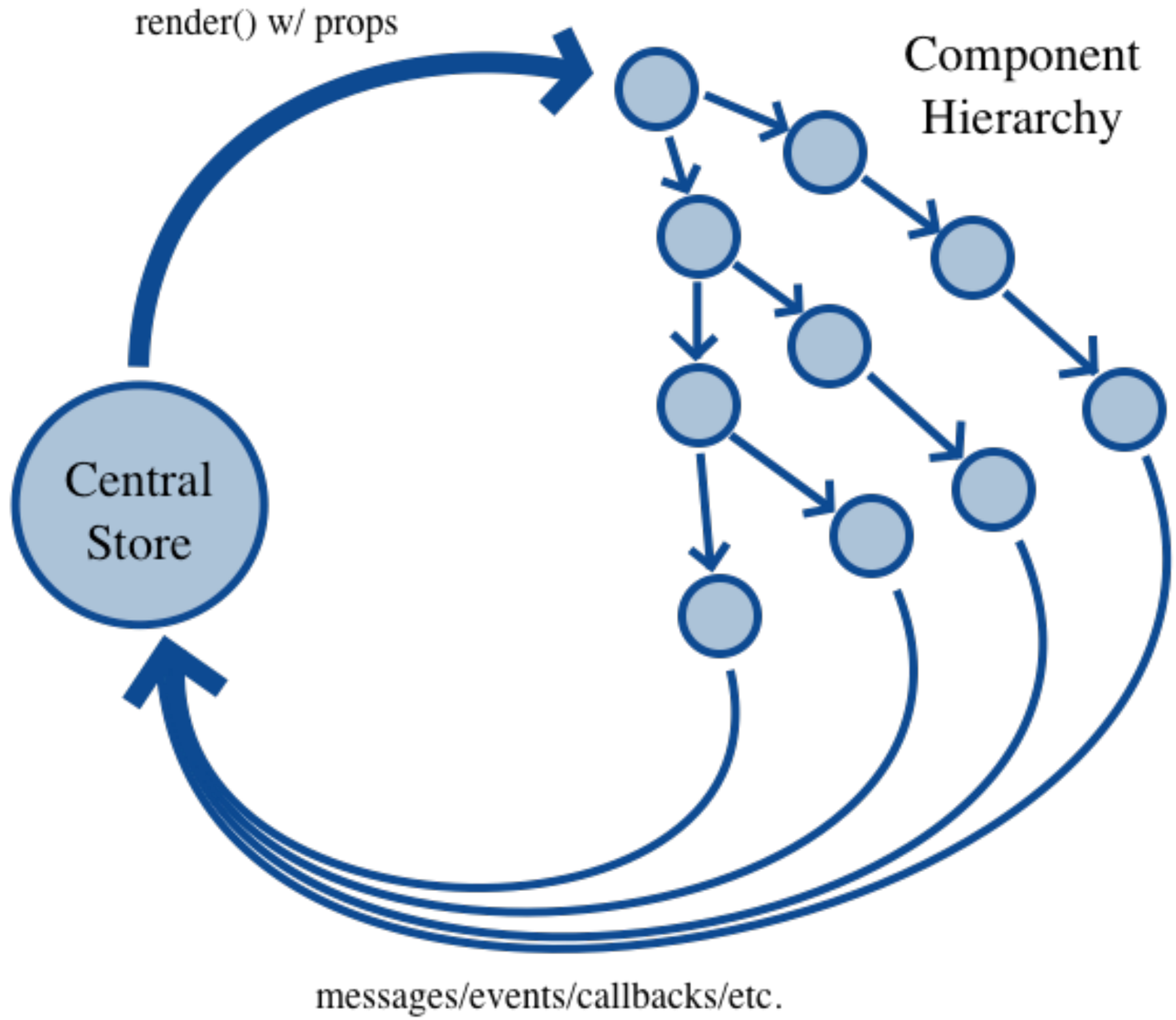
- Cuando se produce el evento sobre el hijo, este llama al *callback* del padre

```
toggle: function() {
    this.props.handleClick(this.props.id);
},
render: function() {
    var atribs = {
      onClick: this.toggle
    };
    if (this.props.comprado) {
      atribs.className = 'tachado';
    }
    return (<li {...atribs}>{this.props.nombre}
            ({this.props.cantidad})</li>);
}
```


© 2015-2016 Depto. Ciencia de la Computación e IA

¿Por qué todo tan complicado?

- Todo es por reducir al máximo el número de componentes con estado
- Esto simplifica el flujo de datos



© 2015-2016 Depto. Ciencia de la Computación e IA

Índice

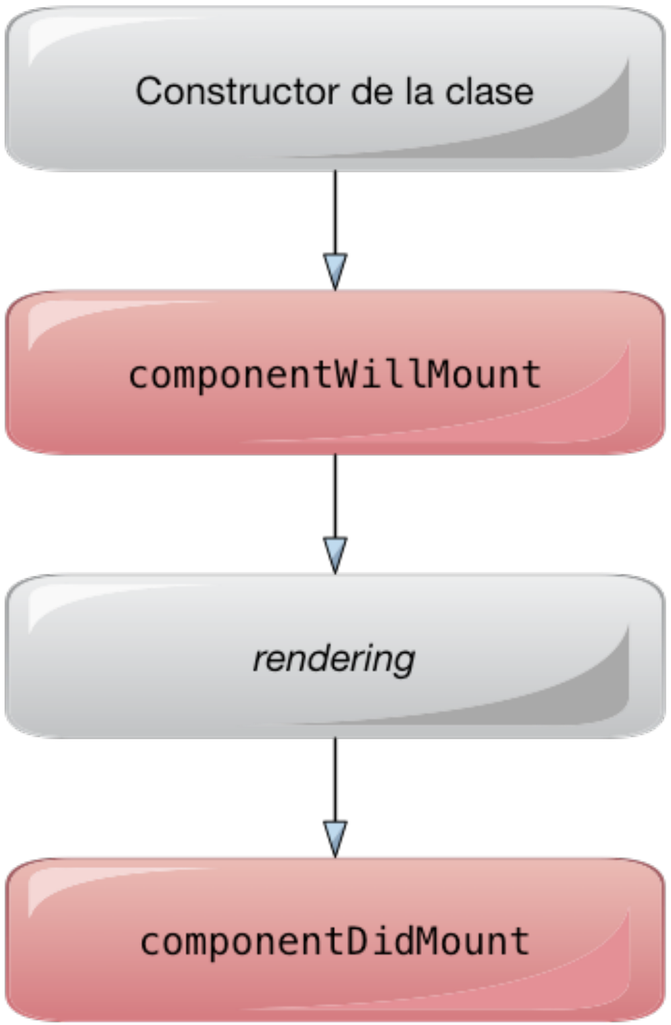
© 2015-2016 Depto. Ciencia de la Computación e IA

Fases del ciclo de vida

React nos da un conjunto de *books* o métodos que podemos implementar y que se llamarán en ciertos momentos del ciclo de vida del componente

© 2015-2016 Depto. Ciencia de la Computación e IA

Fases del montado de componentes



Llamado una única vez, antes del *rendering* inicial

Llamado una única vez, después del *rendering* inicial. Aquí el componente ya tiene una representación en el DOM. Este método es invocado antes en los hijos que en los padres. Apropiado para integración con otros *frameworks* JS, inicialización de *timers* o solicitud de datos vía AJAX

© 2015-2016 Depto. Ciencia de la Computación e IA

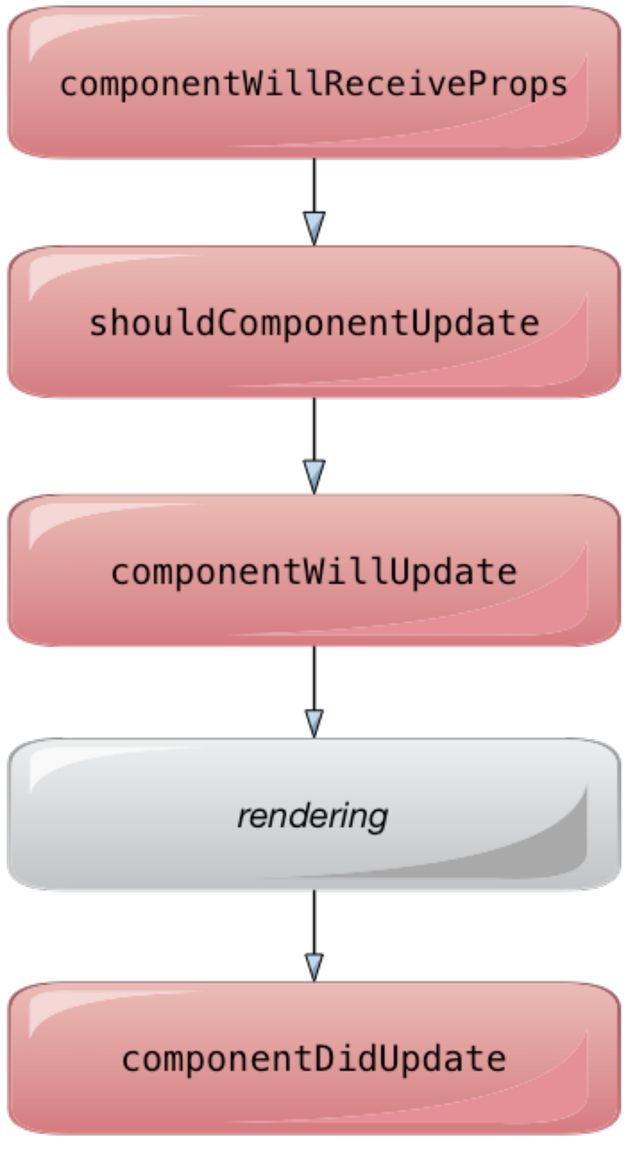
ComponentDidMount para hacer peticiones AJAX

- Si el componente muestra datos obtenidos de un API del servidor

http://jsbin.com/goyozehapu/edit?js,output

```
componentDidMount: function() {
    var peticion = new XMLHttpRequest();
    peticion.onreadystatechange = function() {
      if (peticion.readyState == 4) {
        var datos = JSON.parse(peticion.responseText);
        this.setState({nombre: datos.name, url_avatar:datos.avatar_url});
      }
    }.bind(this);
    peticion.open('GET', 'https://api.github.com/users/'+this.props.login, true);
    peticion.send();
    this.peticion = peticion;
},
componentWillUnmount: function() {
    this.peticion.abort();
},
```

Fases del cambio en props & cambio en state



Llamado cuando el componente recibe nuevos props. Útil para cambiar el estado en función de los nuevos props, pero este cambio no va a disparar un nuevo *rendering*

Si implementamos este método y devolvemos false, indicamos que el cambio no debería suponer un re-renderizado. Útil para mejorar la eficiencia

Invocado inmediatamente antes del *rendering*, aquí no deberíamos cambiar el estado

Invocado inmediatamente después del *rendering*, el DOM del componente ya está actualizado. Útil para hacer operaciones adicionales sobre el DOM

Índice

© 2015-2016 Depto. Ciencia de la Computación e IA

React & Backbone

- React implementa una forma de **mixins**, que nos permiten compartir código Javascript entre múltiples componentes, sin tener que repetirlo.
- Hay implementaciones de terceros de *mixins* para **combinar Backbone y React**.
  - Tener un componente React asociado a 1 o varios modelos/colecciones
  - Vamos a usar aquí una llamada backbone-react-component.
- El mixin sirve de "pegamento" entre componentes React y modelos y/o colecciones de Backbone.
  - Si tenemos un componente React asociado a una colección y esta cambia, el mixin disparará el re-renderizado.

© 2014-2015 Depto. Ciencia de la Computación e IA

Componente con un modelo asociado

- Atributos del modelo accesibles como propiedades de `state`

```
<script type="text/jsx">
    var LibroComp = React.createClass({
        mixins: [Backbone.React.Component.mixin],
        render: function() {
          return (
            <div class="libro">
              <b>{this.state.titulo}</b>, por <em>{this.state.autor}</em>
            </div>
          );
        }
    });
    var libro1 = new LibroModel({titulo:"Crónicas marcianas", autor: "Ray Bradbury"});
    React.render(<LibroComp model={libro1}></LibroComp>,
                 document.getElementById('un_libro'));
</script>
```


© 2014-2015 Depto. Ciencia de la Computación e IA

Componente con un modelo asociado (II)

- El modelo también es accesible con `getModel()`

```
...
render: function() {
  var m = this.getModel();
  return (
    <div class="libro">
      <b>{m.get('titulo')}</b>, por <em>{m.get('autor')}</em>
    </div>
  );
}
...
```


© 2014-2015 Depto. Ciencia de la Computación e IA

Componente con una colección asociada (I)

- Definición de colección y modelos que contiene

```
<script type="text/javascript">
    var LibroModel = Backbone.Model.extend({});
    var Biblioteca = Backbone.Collection.extend({
        model: LibroModel
    });
    var miBiblio = new Biblioteca([
        new LibroModel({titulo: "Juego de tronos", autor: "George R.R. Martin"}),
        new LibroModel({titulo: "El mundo del río", autor: "Philip J. Farmer"})
    ]);
</script>
```


© 2014-2015 Depto. Ciencia de la Computación e IA

Componente con una colección asociada (II)

- Componente "**padre**"

```
<script type="text/jsx">
    var ListaLibros = React.createClass({
        mixins: [Backbone.React.Component.mixin],
        render: function() {
            var libros = this.getCollection().map(function(libro) {
                return (
                    <Libro key={libro.cid} model={libro}/>
                );
            });

            return (
              <div className="listaLibros">
                 {libros}
              </div>
            );
        }
```

Componente con una colección asociada (II)

- Componente "**hijo**"

```
var Libro = React.createClass({
    mixins: [Backbone.React.Component.mixin],
    render: function() {
            return (
              <div className="libro">
                  <b>{this.state.autor}</b>, por <em>{this.state.titulo}</em>
              </div>
            );
    }
 });

React.render(
    <ListaLibros collection={miBiblio}></ListaLibros>,
    document.getElementById('example')
);
```


© 2014-2015 Depto. Ciencia de la Computación e IA

¿Preguntas?

© 2014-2015 Depto. Ciencia de la Computación e IA

¿Preguntas?

© 2015-2016 Depto. Ciencia de la Computación e IA