BackboneJS

Sesión 6 - Introducción a React

Índice

El problema de las interfaces dinámicas

- Hay que estar constantemente redibujando la interfaz cuando cambian modelos
- En Backbone `render()` por convenio **pinta todo el HTML**
- Como Backbone no tiene *data binding* **debemos definir varios `renderXXX()`** para actualizar las diversas partes de la interfaz

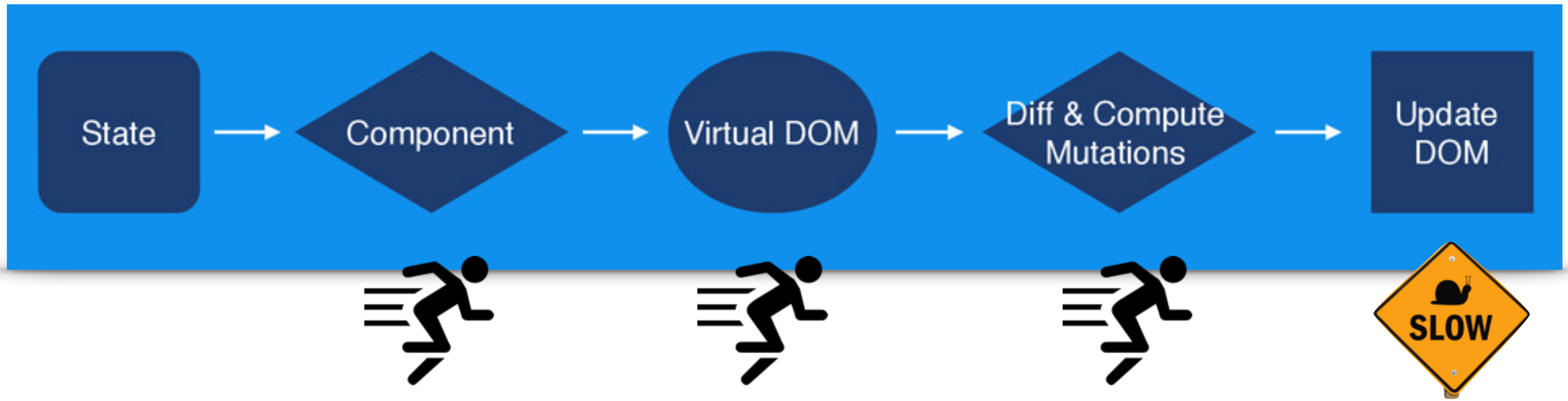
```
render: function() {
        this.$el.html('<input type="text" id="localidad">' +
           '<button>Ver tiempo</button>' +
           '<div> <img id="icono" src=""></div>' +
           '<div id="descripcion"></div>');
},
renderData: function() {
        $('#icono').attr('src',this.model.get("icono_url"));
        $('#descripcion').html(this.model.get("descripcion"));
},
```

Solución radical

¿Por qué no llamar siempre a `render()` cada vez que cambie cualquier dato? así estaremos seguros de que la vista está actualizada, y el código se mantiene simple.

React

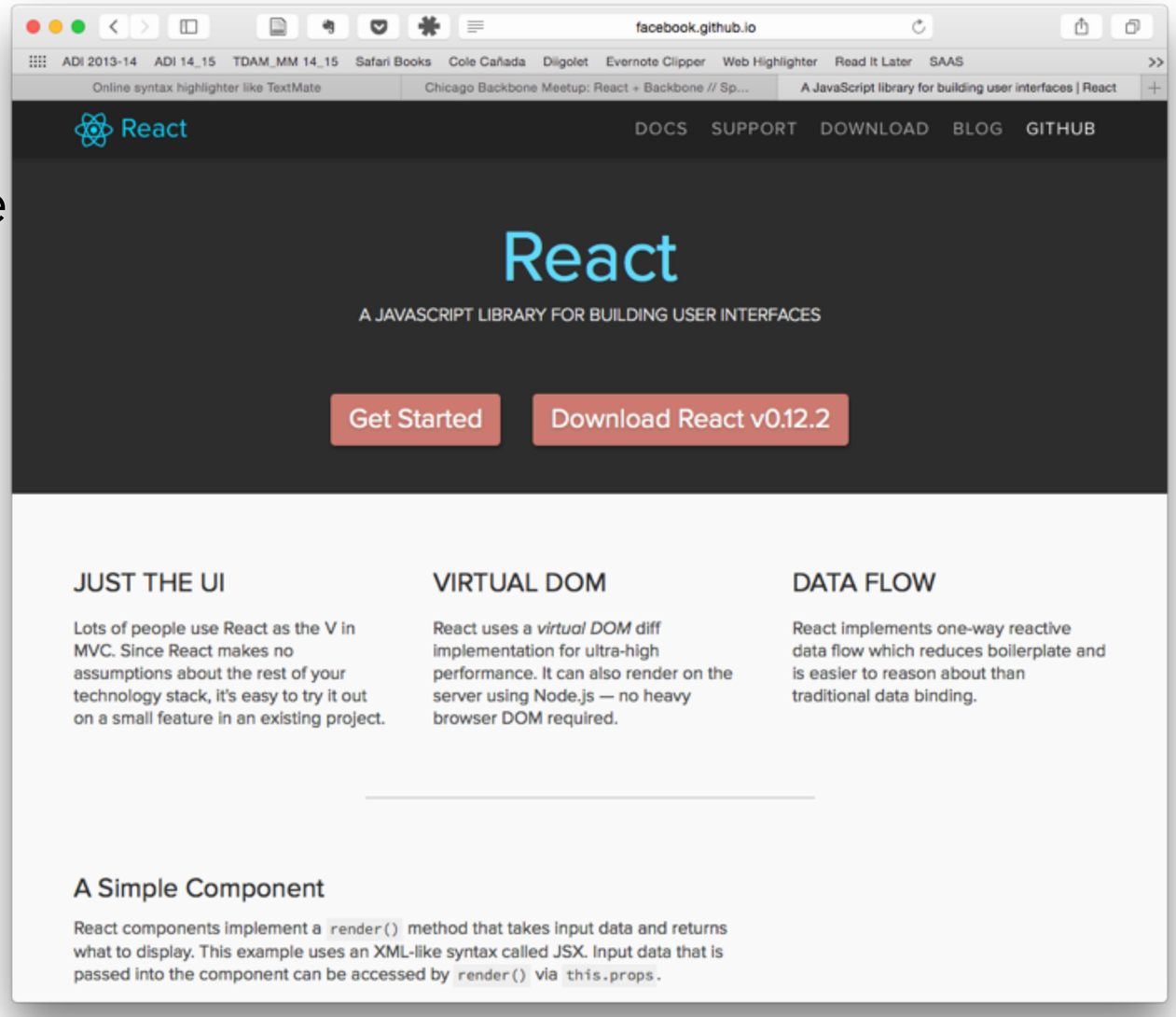
- Hace una especie de `diff` entre la nueva vista y la vista actual, y solo actualiza lo necesario
- Trabaja internamente con un “DOM virtual”, sobre el que hace las operaciones, y solo al final actualiza el DOM real



¿De dónde sale React?

- Desarrollado en Facebook
- Usado en producción en multitud de sitios “grandes”: Facebook, Instagram, AirBnb, Khan Academy,…

http://facebook.github.io/react/



React & MVC

- React **no es un framework MVC**, solo sirve para hacer interfaces
- Podemos usarlo con MVC
  - Podemos sustituir las vistas de Backbone por componentes de React, y seguir usando modelos y colecciones de Backbone
  - Facebook usa “su propia” arquitectura, en lugar de MVC: **Flux**

Índice

Componentes React

- Encapsulan el **lenguaje de marcado** junto con la lógica de presentación y el **manejo de eventos**

```
<div id="componente"></div>
<script>
  var comp = React.DOM.h1({id:"saludo"}, '¡Hola React!');
  ReactDOM.render(comp, document.getElementById('componente'));
</script>
```

Clase componente

- Normalmente **crearemos una clase** para encapsular el componente

http://jsbin.com/mikudodovi/edit?html,js,output

```
<script src="lib/react.js"></script>
<script src="lib/react-dom.js"></script>
<div id="miApp"></div>
<script>
    var MiPrimerComponente = React.createClass({
      render: function() {
        var mensaje = "Soy un componente personalizado";
        mensaje +=    " (" + mensaje.length + " caracteres )";
        return React.DOM.h1(null, mensaje);
      }
    });
    var instancia = React.createElement(MiPrimerComponente, null);
    ReactDOM.render(instancia, document.getElementById('miApp'));
</script>
```

Índice

Props

- Con ellas le pasamos información al componente desde “el exterior”
- Aunque se pueden modificar con `setProps()`, **no deberían modificarse una vez instanciado el componente**

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

```
var MiComponente = React.createClass({
      render: function() {
        var mensaje = this.props.texto;
        mensaje +=   " (" + mensaje.length + " caracteres )";
        return React.DOM.h1(null, mensaje);
      }
    });
var instancia = React.createElement(MiComponente, {texto:'Hola React’});
ReactDOM.render(instancia, document.getElementById('miApp'));
```

Iterando por las props

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

```
var ListaCompra = React.createClass({
      render: function() {
        return React.DOM.ul(null,
          this.props.items.map(function(item) {
            var texto = item.nombre + ' (' + item.cantidad + ')';
            return React.DOM.li(null, texto);
          })
        );
      }
    });
var lista = [{nombre:'huevos', cantidad:12}, {nombre:'pan', cantidad:1}];
var instancia = React.createElement(ListaCompra, {items: lista});
ReactDOM.render(instancia, document.getElementById('miApp'));
```

Listas de nodos y el atributo key

- Cuando tenemos un conjunto de nodos del mismo tipo debemos identificarlos con una key única

```
var ListaCompra = React.createClass({
  render: function() {
    return React.DOM.ul(null,
      this.props.items.map(function(item, pos) {
        var texto = item.nombre + ' (' + item.cantidad + ')';
        return React.DOM.li({key:pos}, texto);
      })
    );
  }
});
```

Índice

Estado

- Se guarda en una propiedad `state` del componente, que no es más que un objeto JS
- A diferencia de `props`, el estado está pensado para modificarse durante el ciclo de vida, se hace con `setState`
  - Aunque podríamos modificar `state` directamente, es mejor hacerlo a través del API. React "acumula" los cambios de estado y los hace más eficientes
- Especificamos el estado inicial en una función `getInitialState`

Ejemplo lista de la compra con estado

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

Para probarlo, cambiar el estado desde la consola: `comp.setState({comprados:[true,false]})`

Índice

Eventos en JS/HTML “clásico”

```
<p onClick="alert('has hecho clic')">Hola</p>
```

Eventos en React

- Muy similar, al menos en apariencia

```
var MiComponente = React.createClass({
    saludar: function() {
      alert('Has hecho clic');
    },
    render: function() {
      return React.DOM.h1({onClick:this.saludar}, 'Hola');
    }
  });
var instancia = React.createElement(MiComponente, null)
ReactDOM.render(instancia, document.getElementById('miApp'));
```

I Am Devloper
@iamdevloper

Seguir

1995: just do onclick='myFunc()'

2010: do that and you are fired. events declared in JS files only plz

2015: Oh, React uses onClick...cool.

Eventos React vs. "clásicos"

- React **estandariza los eventos y sus propiedades**, para que sean portables entre navegadores
  - Los eventos que manejamos en código React no son directamente los del navegador, sino una capa de abstracción por encima, al estilo jQuery

- React usa **delegación de eventos**, es decir no coloca el *listener* físicamente donde lo ponemos nosotros, sino más arriba en la jerarquía del DOM

Ejemplo “lista de la compra” completo

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

Índice

El problema del React.DOM.* en el render()

- Al empezar a meter etiquetas dentro de etiquetas, el código se va haciendo más tedioso
- El mismo problema que tenemos con el API del DOM estándar

```
var miLista = React.DOM.ul(null,
              React.DOM.li(null, 'Pan'),
              React.DOM.li(null, 'Patatas')
    );
ReactDOM.render(miLista, document.getElementById('miApp'));
```

JSX al rescate

- JSX permite introducir HTML (en realidad XML) en el código JS , y por tanto expresar nuestro componente en un formato similar al que tendrá en la página

```
var miLista = <ul> <li>Pan</li> <li>Patatas</li> </ul>
ReactDOM.render(miLista, document.getElementById('miApp'));
```

- No solo podemos meter HTML en el JS, también JS dentro del HTML

```
var saludo = "Hola";
var miComponente = <h1 id="saludo">¡{saludo} React!</h1>;
ReactDOM.render(miComponente,document.getElementById('miApp'));
```

Un ejemplo un poco más complicado de JSX

- Con JSX "creamos" nuestras propias etiquetas

```
var ListaCompra = React.createClass({
      render: function() {
        var items = this.props.items.map(function(item, indice){
          return <li key={indice}> {item.nombre} ({item.cantidad}) </li>
        });
        return <ul>{items}</ul>
      }
});

var lista = [{nombre:'huevos', cantidad:12}, {nombre:'pan', cantidad:1}];
var instancia = <ListaCompra items={lista}/>;
var componente = ReactDOM.render(instancia, document.getElementById('miApp'));
```

"Gotchas" de JSX

- JSX usa sintaxis XML, no HTML

```
//¡¡no va a funcionar!!
return <a onclick={this.saludar} href='#'>Clicame</a>;
//pero esto sí
return <a onClick={this.saludar} href='#'>Clicame</a>;
```

- `render()` debe devolver una única etiqueta de "nivel superior"

```
//NOOOOOOOO!!!!!!
return (
  <span>Hola</span>
  <span>mundo</span>
)
```

JSX no es JS

- El código JSX **no es directamente interpretable** por el navegador
- Necesitamos un **transpilador** que traduzca de JSX a JS

```
// Original (JSX):
var app = <div className="pie" />;
// Resultado (JS):
var app = React.createElement('div', {className:"pie"});
```

- El transpilador más conocido es **Babel** (http://babeljs.io). No solo traduce de JSX a JS, sino también de ES6 a ES5
- ¿**Cuándo** se hace la transpilación?
  - En una fase previa, para apps en producción
  - Sobre la marcha, dentro del propio navegador. Apropiado para pruebas.

Transpilación “sobre la marcha”

- Incluir Babel para navegador (la versión actual, la 6, ya no tiene esto)

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/
browser.min.js"></script>
```

- Poner el código con `type=“text/babel”`

```
<script type="text/babel">
    var saludo = "Hola";
    var miComponente = <h1 id="saludo">¡{saludo} React!</h1>;
    ReactDOM.render(miComponente,document.getElementById('miApp'));
</script>
```

Transpilación previa

- En *frontend* existen herramientas para construir el proyecto al igual que en backend está Maven
- Babel se puede usar con multitud de herramientas de *workflow* para *frontend*
- *En el apéndice de los apuntes tenéis una descripción del uso de Babel junto con Browserify y npm*

¿Preguntas?