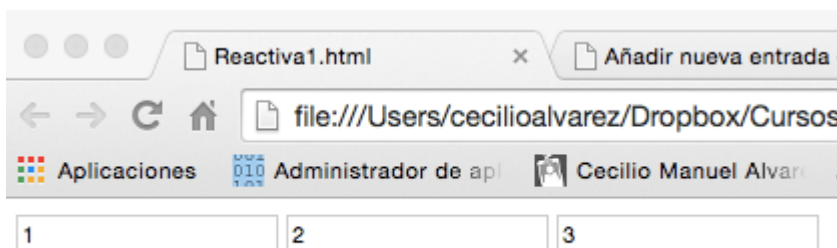


JavaScript Reactive Programming es uno de los conceptos que empieza a estar de moda, lamentablemente no son concepto sencillos de entender y cuesta hacerse a la idea de como funciona este nuevo paradigma. Vamos a intentar explicar los conceptos de programación reactiva a través de un ejemplo.



Tenemos tres cajas de texto, en las dos primeras cajas escribimos un par de números y en la última se calcula la suma. Se trata de un ejemplo muy similar a una hoja de cálculo clásica. Vamos a implementar esta funcionalidad utilizando jQuery:

```
<html>
<head>
<script src="jquery-1.11.1.js"></script>
<script type="text/javascript">

$(document).ready(function() {

var suma=0;

$("#caja1").keyup(function() {
suma=parseInt($("#caja2").val())+parseInt($(this).val());
```

```

$("#resultado").val(suma);

});
$("#caja2").keyup(function() {

suma=parseInt($("#caja1").val())+parseInt($("#this").val());

$("#resultado").val(suma);
});
});

</script>

</head>

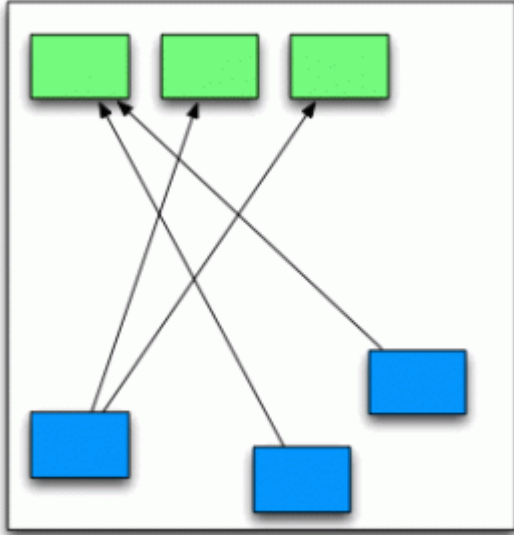
<body>

<input type="text" id="caja1" value="0">
<input type="text" id="caja2" value="0"/>
<input type="text" id="resultado">
</body>
</html>
<p class="p1"></p>

```

```
<pre>
```

El código es bastante fácil de entender, simplemente ligamos los eventos “keyup” a cada una de las cajas y realizamos los cálculos . El problema que tenemos es que estamos obligados a mantener una variable con el estado de la suma. No es un gran problema ... pero no lo es cuando abordamos un ejemplo sencillo. Si tuviéramos que realizar esta actualización de estados en muchas partes del documento HTML el número de variables aumentaría y el código se convertiría en inmanejable.



¿Qué es JavaScript Reactive Programming?

La programación reactiva es una programación orientada a gestionar flujos de información de forma asíncrona. Esto de entrada nos puede resultar complejo de entender pero básicamente estamos hablando de programación funcional asíncrona. Vamos a construir el mismo bloque de código pero utilizando **Bacon.js** una librería de programación reactiva de JavaScript.

```
</p>
<html>
<head>
<script src="jquery-1.11.1.js"></script>
<script src="bacon.js"></script>
<script type="text/javascript">

$(document).ready(function() {

flujoCajal=$("#cajal").asEventStream("keyup");
```

```
flujoCaja2=$("#caja2").asEventStream("keyup");
flujoCaja1
.merge(flujoCaja2)
.map('.target.value')
.scan(0,function(x,y) {

return parseInt(x)+parseInt(y);

}).assign($("#resultado"),"val");

});
```

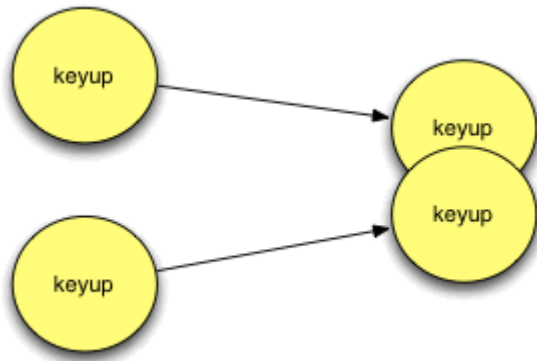
```
</script>
```

```
</head>
```

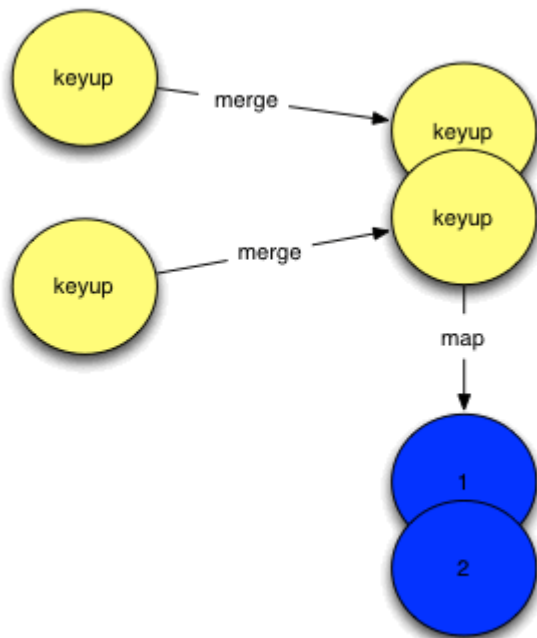
```
<body>
```

```
<input type="text" id="caja1" value="0">
<input type="text" id="caja2" value="0"/>
<input type="text" id="resultado">
</body>
</html>
<p class="p1">
```

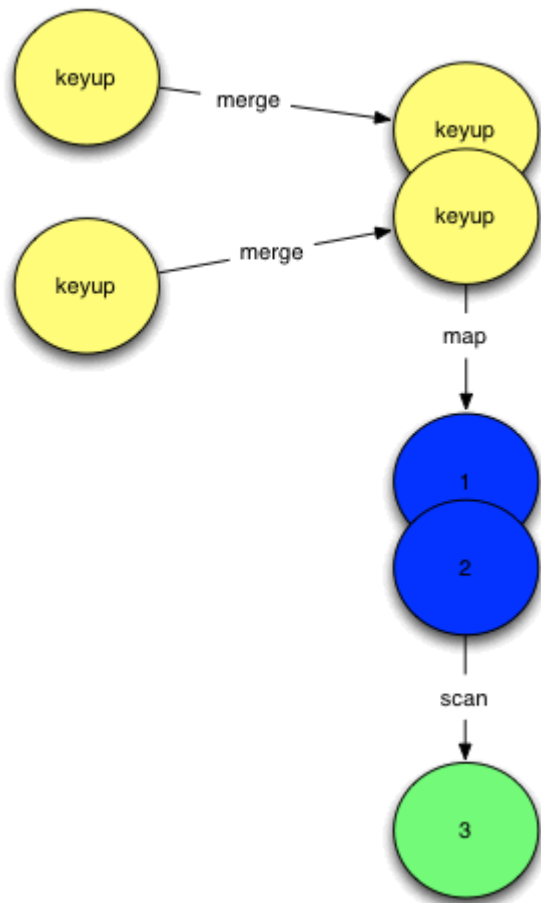
El código es muy compacto y mucho más expresivo que el construido con jQuery. ¿Cómo funciona exactamente ?. El primer paso que se realiza es construir dos flujos de eventos (“keyup”) y agruparlos con la instrucción merge.



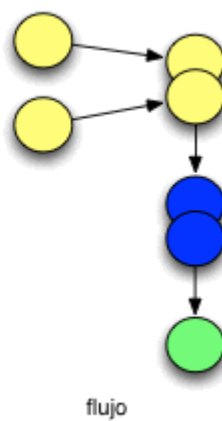
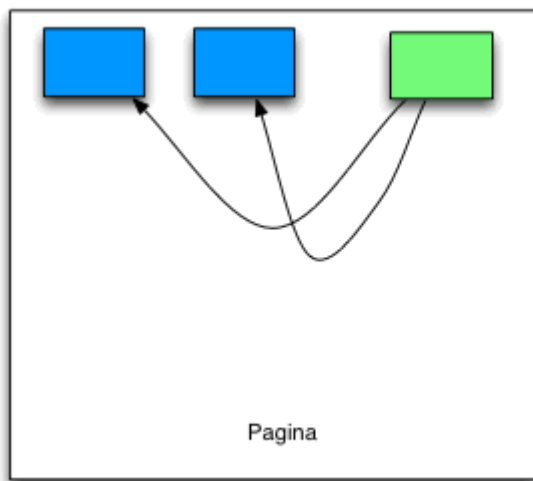
Realizada esta operación el siguiente paso es usar la función `map()` que realiza una conversión de los eventos a los datos concretos que necesitamos de los eventos. En nuestro caso nos queremos quedar con el valor de cada una de las cajas , de ahí que usemos `".target.value"` .



Una vez tengamos los datos en un único flujo usamos la función `scan` que es la encargada de recorrerlos y sumarlos todos.



Por último usamos la función de `assign` para asignar el resultado almacenado por `scan` a la caja de resultado. Estamos creando un flujo de trabajo con programación funcional asíncrona algo que cada día las páginas web necesitan más.



Estas tecnologías poco a poco se irán haciendo un fuerte hueco en el mercado.

Otros artículos relacionados [JavaScript Map y JSON](#) , [jQuery Sizzle](#)