Programación funcional

Programación funcional

La vamos a entender como:

- Creación y manipulación de funciones
- Alteración de funciones
- Aplicación de funciones
- Asincronía

Funciones de orden superior

Funciones que devuelven funciones.

- curry
- bind
- ¡Y muchas otras!

Funciones de orden superior

Algunas de las más útiles:

- throttle
- debounce
- once
- after
- compose
- memoize

Funciona - throttle

Controla la frecuencia de invocación de una función durante un tiempo determinado.

La función se invocará solamente 1 vez cada n milisegundos.

Funcional - throttle

```
var counter = 0,
    inc = function() { counter++; };

var incWrapper = throttle(inc, 10);

for (var i=100000; i--;) {
  incWrapper();
}

alert(counter); // ~6
```

Funcional - throttle

```
function throttle(fn, time) {
  var last = 0;
  return function() {
    var now = new Date();
    if ((now - last) > time) {
      last = now;
      return fn.apply(this, arguments);
    }
  }
}
```

Funcional - debounce

Ejecuta la función n milisegundos después de haber sido llamada.

Desde la última invocación.

Funcional - debounce

```
var counter = 0,
    inc = function() {
       counter++;
       alert(counter);
    };

inc = debounce(inc, 1000);

for (var i=100000; i--;) {
  inc();
}
```

redradix

Funcional - debounce

```
function debounce(fn, time) {
  var timerId;
  return function() {
    var args = arguments;

    var fnz = function() {
      fn.apply(this, args);
    }
    if (timerId) clearTimeout(timerId);
    timerId = setTimeout(fnz.bind(this), time);
  }
}
```

Funcional - once

Función que se puede aplicar solo una vez

```
var counter = 0,
    inc = function() {
        counter++;
        alert(counter);
    };

inc = once(inc);

for (var i=100000; i--;) {
  inc();
}
```

redradix

Funcional - once

```
function once(fn) {
  var executed = false;
  return function() {
    if (!executed) {
      executed = true;
      return fn.apply(this, arguments);
    }
  }
}
```

Funcional - after

Función que se ejecuta tras haber sido llamada n veces

```
var counter = 0,
    inc = function() {
       counter++;
    };

inc = after(inc, 1000);

for (var i=100000; i--;) {
  inc();
}

alert(counter);
```

redradix

Funcional - after

```
function after(fn, n) {
  var times = 0;
  return function() {
    times++;
    if (times % n == 0) {
      return fn..call(this, ...arguments);
    }
  }
}
```

Funcional - compose

Composición de funciones.

```
function multiplier(x) {
  return function(y) { return x*y; }
}

var randCien = compose(Math.floor,
 multiplier(100),
 Math.random);
alert(randCien(5));
```

Funcional - compose

```
function compose() {
  var fns = [].slice.call(arguments);
  return function(x) {
    var currentResult = x, fn;
    for (var i=fns.length; i--;) {
      fn = fns[i];
      currentResult = fn(currentResult);
    }
    return currentResult;
  }
}
```

Funcional - memoize

Nunca calcules el mismo resultado 2 veces!

- La primera invocación calcula el resultado
- Las siguientes devuelven el resultado almacenado
- Solo vale para funciones puras

Funcional - memoize

```
function fact(x) {
  if (x == 1) { return 1; }
  else { return x * fact(x-1); }
}

const factWrapper = memoize(fact);

var start = new Date();
factWrapper(100);
console.log(new Date() - start);
start = new Date();
factWrapper(100);
console.log(new Date() - start);
```

Funcional - memoize

```
function memoize(fn) {
  var cache = {};
  return function(p) {
    var key = JSON.stringify(p);
    if (!(key in cache)) {
      cache[key] = fn.apply(this, arguments);
    }
    return cache[key];
  }
}
```

redradix