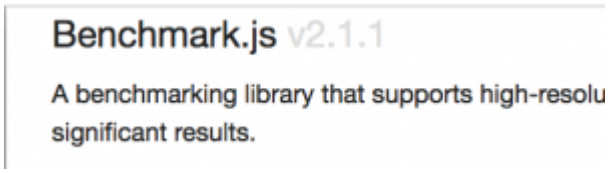
La necesidad de crear JavaScript Benchmarks es cada día más obligatoria, ya que el código de JavaScript que necesitamos construir es cada día mayor. Hay que tener en cuenta que ademas JavaScript es uno de los lenguajes que más particularidades tiene. Así pues es fácil encontrarnos con situaciones que necesiten ejecutar pruebas de rendimiento. Vamos a introducir la herramienta Benchmarks.js para realizar unos test de rendimiento apoyándonos en jQuery.



JavaScript BenchMarks

El primer paso es descargarnos BenchMark.js LoDash.js y jQuery.js que son las librerías de JavaScript que vamos a utilizar. Una vez hecho esto vamos a construir un bloque de código que nos genere mil filas de una tabla con jQuery.

```
for (var i=0;i<1000;i++) {

 $("table").append("<tr><td>hola</td><td>hola</td></tr>");

 }
```

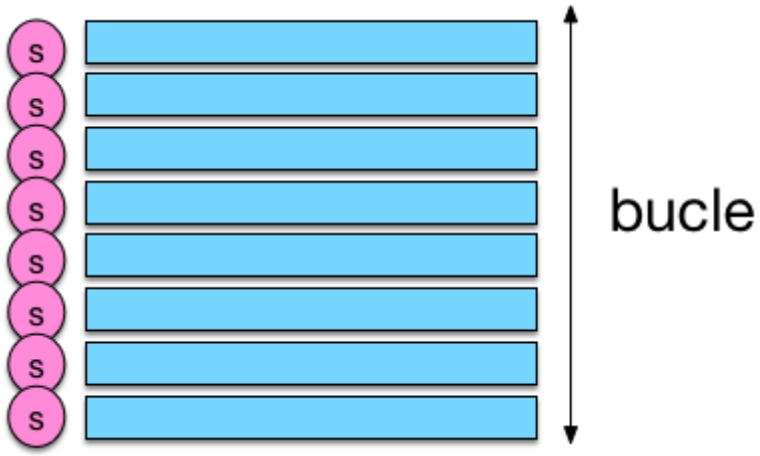
El código funciona correctamente pero no es muy óptimo. Vamos a ejecutar este código junto con un par de modificaciones sobre el utilizando Benchmark.js.

```
var suite = new Benchmark.Suite;

 suite.add('clasico', function() {
```

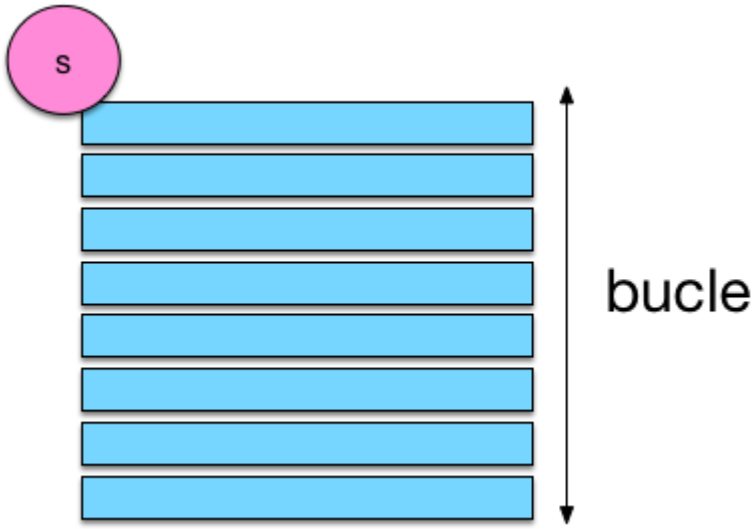
```
for (var i=0;i<1000;i++) {

$("table").append("<tr><td>hola</td><td>hola</td></tr>");

}

}).add('cache selector', function() {

var tabla=$("table");
for (var i=0;i<1000;i++) {

tabla.append("<tr><td>hola</td><td>hola</td></tr>");

}

}).
add('no redibujar', function() {

var tabla=$("table");

var texto="";

for (var i=0;i<1000;i++) {

texto+=("<tr><td>hola</td><td>hola</td></tr>");

}

tabla.append(texto);

})
```

```
 .on('cycle', function(event) {
 console.log(String(event.target));
 }).on('error', function(mensaje) {
 console.log(mensaje);
 }).on('complete', function() {
 console.log('el mas rapido es: ' +
this.filter('fastest').map('name'));
 }).run();
```

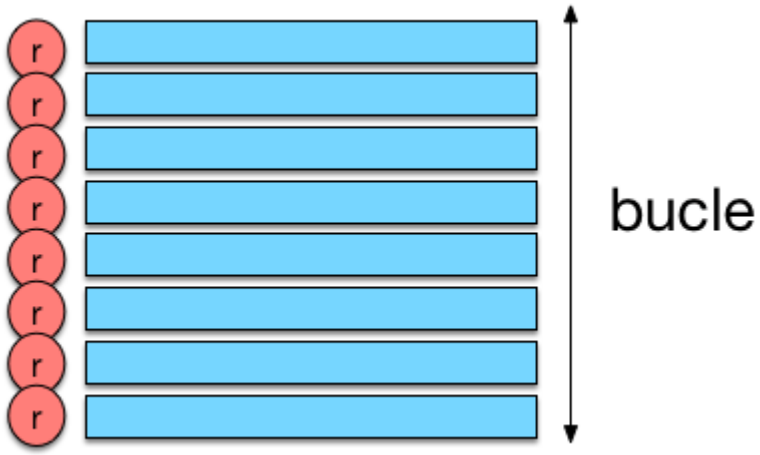
Hemos creado tres test diferentes: clásico ,cache selector , y no redibujar. El clásico tiene un problema muy importante y es que cada vez que entramos en una iteración del bucle se genera un nuevo selector.



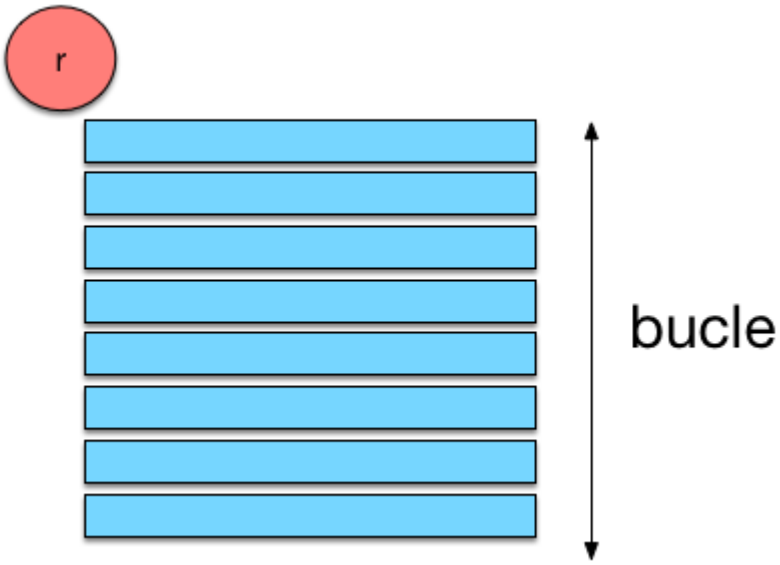
El segundo se denomina cache selector y se encarga de solventar el problema que tiene el código. Cachea el selector de jQuery que se construye fuera del bucle.



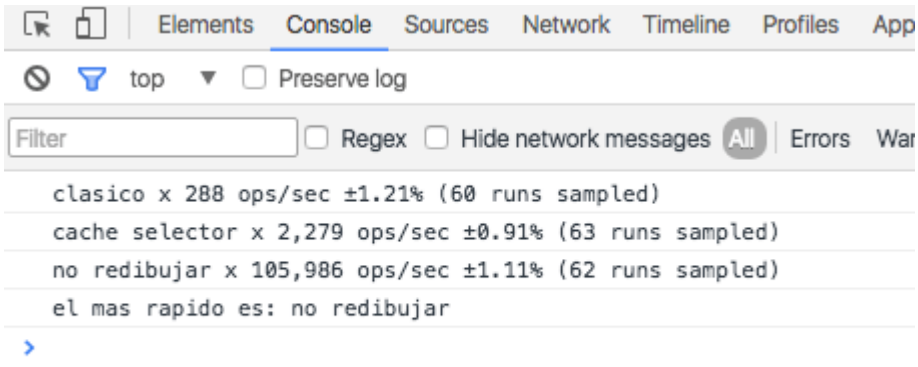
El tercer caso extrae del bucle la iteración principal:



Evitando que se tenga que redibujar la página cada vez que hacemos una iteración.



Es momento de ejecutar esta página de Test y ver como la librería de Benchmarking se encarga de decirnos cual tiene un rendimiento mejor.



Como podemos ver de una forma sencilla nos deja claro que el de no redibujar es el que mayor rendimiento tiene ya que incluye las mejoras que existían en cache selector. En muchas ocasiones código de JavaScript que puede parecer correcto esconde errores importantes.

Otros artículos relacionados: JavaScript ES6 API , JavaScript Streams vs Promises , JavaScript Pure Functions