Measure and count executions

By Meggin Kearney
Meggin is a Tech Writer

By Flavio Copes
Flavio is a Full Stack Developer

By Paul Bakaus
Open Web Developer Advocate at Google • Tools, Performance, Animation, UX

Take advantage of the Console API to measure execution times and count statement executions.

TL;DR

- Use `console.time()` and `console.timeEnd()` to track time elapsed between code execution points.
- Use `console.count()` to count how many times the same string is passed to a function.

Measure execution times

The `time()` method starts a new timer and is very useful to measure how long something took. Pass a string to the method to give the marker a name.

When you want to stop the timer, call `timeEnd()` and pass it the same string passed to the initializer.

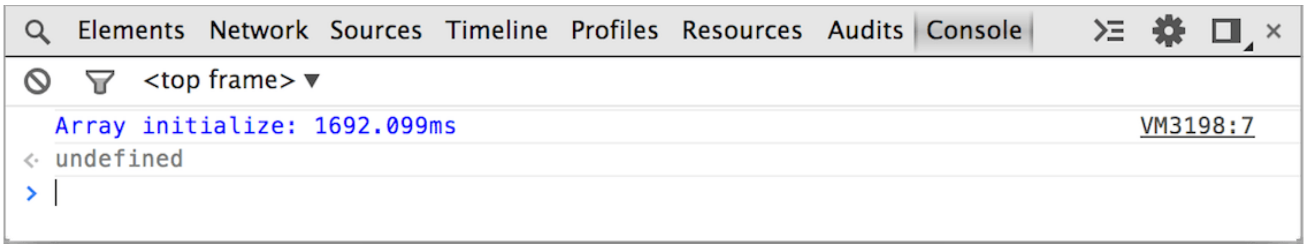
The console then logs the label and time elapsed when the `timeEnd()` method fires.

Basic example

Here, we measure the initialization of a million new Arrays:

```
console.time("Array initialize");
var array= new Array(1000000);
for (var i = array.length - 1; i >= 0; i--) {
    array[i] = new Object();
};
console.timeEnd("Array initialize");
```

Which outputs the following in the Console:

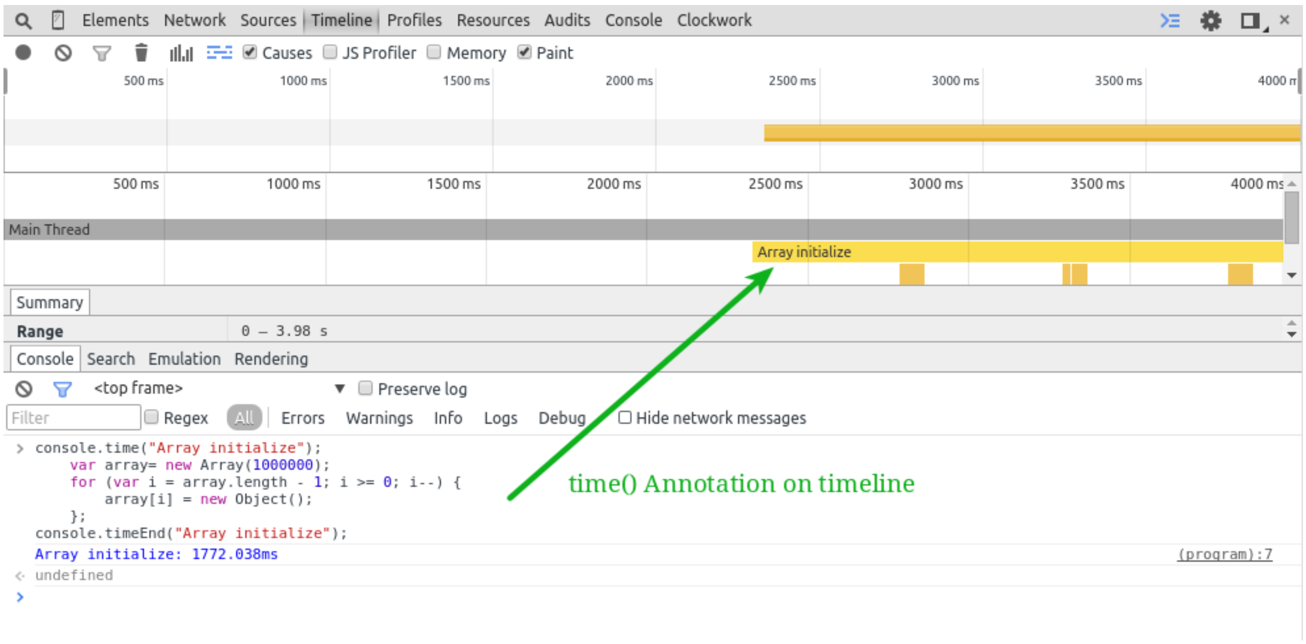


```
Array initialize: 1692.099ms                                    VM3198:7
undefined
```

Timers on the Timeline

When a Timeline recording is taking place during a `time()` operation, it annotates the timeline as well. Use it when you want to trace what your application does and where it comes from.

How an annotation on the timeline looks from `time()`:



```
> console.time("Array initialize");
      var array= new Array(1000000);
      for (var i = array.length - 1; i >= 0; i--) {
          array[i] = new Object();
      };
  console.timeEnd("Array initialize");
  Array initialize: 1772.038ms                                  (program):7
< undefined
>
```

Marking the Timeline

Note: The `timeStamp()` method only functions while a Timeline recording is in progress.

The Timeline panel provides a complete overview of where the engine spends time. You can add a mark to the timeline from the console with the `timeStamp()`. This is a simple way to correlate events in your application with other events.

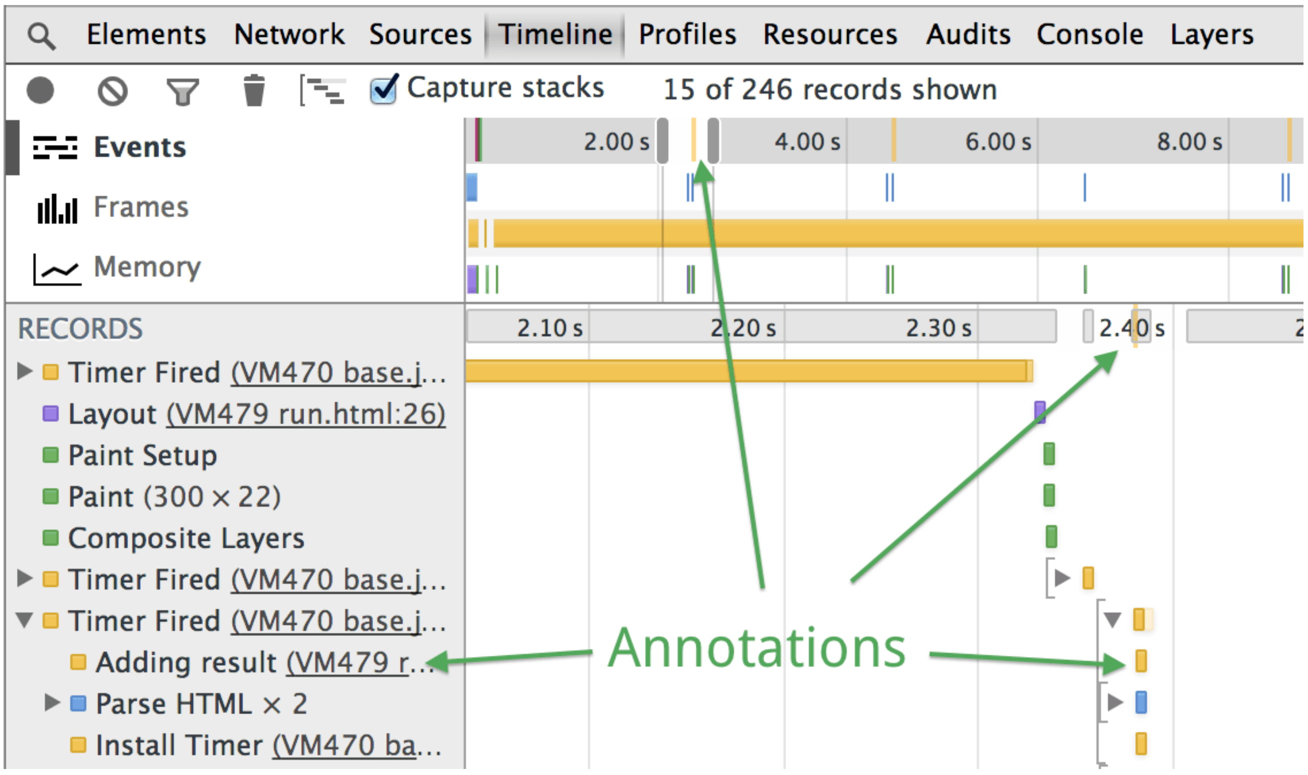
The **timeStamp()** annotates the Timeline in the following places:

- A yellow vertical line in the Timeline's summary and details view.
- It adds a record to the list of events.

The following example code:

```
function AddResult(name, result) {
    console.timeStamp("Adding result");
    var text = name + ': ' + result;
    var results = document.getElementById("results");
    results.innerHTML += (text + "<br>");
}
```

Results in the following Timeline timestamps:



Counting statement executions

Use the **count()** method to log a provided string along with the number of times the same string has been provided. When the exact statement is given to **count()** on the same line, the number is incremented.

Example code of using **count()** with some dynamic content:

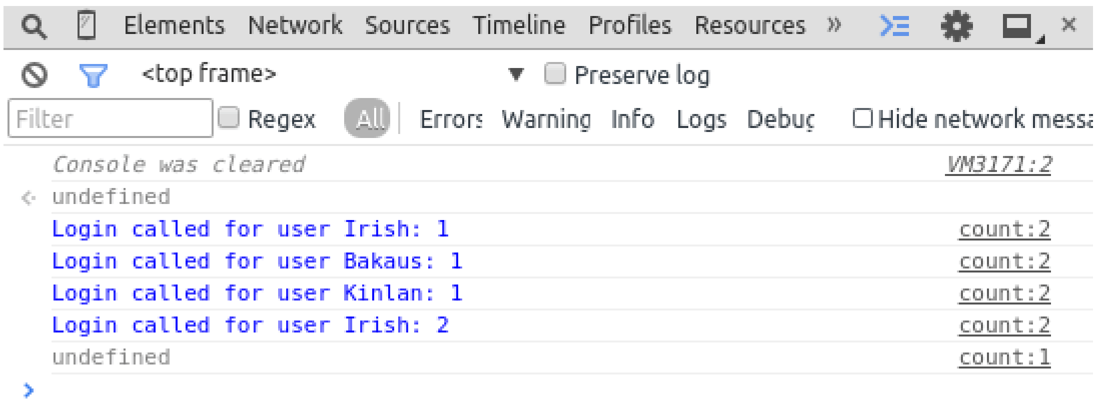
```
function login(user) {
    console.count("Login called for user " + user);
}

users = [ // by last name since we have too many Pauls.
    'Irish',
    'Bakaus',
    'Kinlan'
];

users.forEach(function(element, index, array) {
    login(element);
});

login(users[0]);
```

Output of the code sample:



```
Console was cleared                                    VM3171:2
undefined
Login called for user Irish: 1                          count:2
Login called for user Bakaus: 1                         count:2
Login called for user Kinlan: 1                         count:2
Login called for user Irish: 2                          count:2
undefined                                               count:1
>
```

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 3.0 License, and code samples are licensed under the Apache 2.0 License. For details, see our Site Policies. Java is a registered trademark of Oracle and/or its affiliates.

Last updated July 2, 2018.