عنوان: **آموزش فایرباگ - #4 - JavaScript Development**
نویسنده: احمد احمدی
تاریخ: ۱۳۹۱/۰۵/۰۶ ۱۹:۵۹
آدرس: www.dotnettips.info
برچسب‌ها: FireBug

در قسمت قبل با توابع خط فرمان آشنا شدیم . در این قسمت با توابع کنسول آشنا خواهیم شد .

فایرباگ یک متغییر عمومی به نام console دارد که به همه‌ی صفحات باز شده در فایرفاکس اضافه می‌کند . این شیء متدهایی دارد که بوسیله آن‌ها می‌توانیم عملیاتی در برنامه مان انجام داده و اطلاعاتی را در کنسول چاپ کنیم .

بعضی از این متدها عملکردی مشابه متدهای خط فرمان (که در قسمت قبل شرح داده شدند ،) دارند که از توضیح مجدد آن‌ها اجتناب می‌کنیم .

توابع کنسول - Console API :

توجه : همانند قسمت قبل ، در این قسمت هم برای همراه شدن با تست‌ها ، کد صفحه‌ی زیر را ذخیره کنید و برای اجرای کدها ، آن‌ها را در قسمت خط فرمان (در تب کنسول) قرار بدهید و دکمه‌ی Run (یا Ctrl + Enter) را بزنید .

```
<input type="button" onclick="startTrace('Some Text')" value="startTrace" />
<input type="button" onclick="startError()" value="test Error" />

<script type="text/javascript">
    function startTrace(str) {
        return method1(100, 200);
    }
    function method1(arg1, arg2) {
        return method2(arg1 + arg2 + 100);
    }
    function method2(arg1) {
        var var1 = arg1 / 100;
        return method3(var1);
    }
    function method3(arg1) {
        console.trace();
        var total = arg1 * 100;
        return total;
    }

    function testCount() {
        // do something
        console.count("testCount() Calls Count .");
    }

    function startError() {
        testError();
    }

    function testError() {
        var errorObj = new Error();
        errorObj.message = "this is a test error";
        console.exception(errorObj);
    }

    function testFunc() {
        var t = 0;
        for (var i = 0; i < 100; i++) {
            t += i;
        }
    }
</script>
```

([...,console.log(object[,object

این دستور یک پیغام در کنسول چاپ می‌کند .

```
console.log("This is a log message!");
```

نتیجه :

```
>>> console.log("This is a log message!");
This is a log message!
```

این دستور را می‌توانیم به شکل‌های مختلفی فراخوانی کنیم .
مثلا :

```
console.log(1 , "+" , 2 , "=", (1+2));
```

نتیجه :

```
>>> console.log(1 , "+" , 2 , "=", (1+2));
1 + 2 = 3
```

در این دستور می‌توانیم از چند حرف جایگزین هم استفاده کنیم .

Pattern	Type
%s	String
%d, %i	Integer (numeric formatting is not yet supported)
%f	Floating point number (numeric formatting is not yet supported)
%o	Object hyperlink
%c	Style formatting

مثال :

```
console.log("Firebug 1.0 beta was %s in December %i.","released",2006);
```

نتیجه :

```
>>> console.log("Firebug 1.0 beta was %s in December %i.","released",2006);
Firebug 1.0 beta was released in December 2006.
```

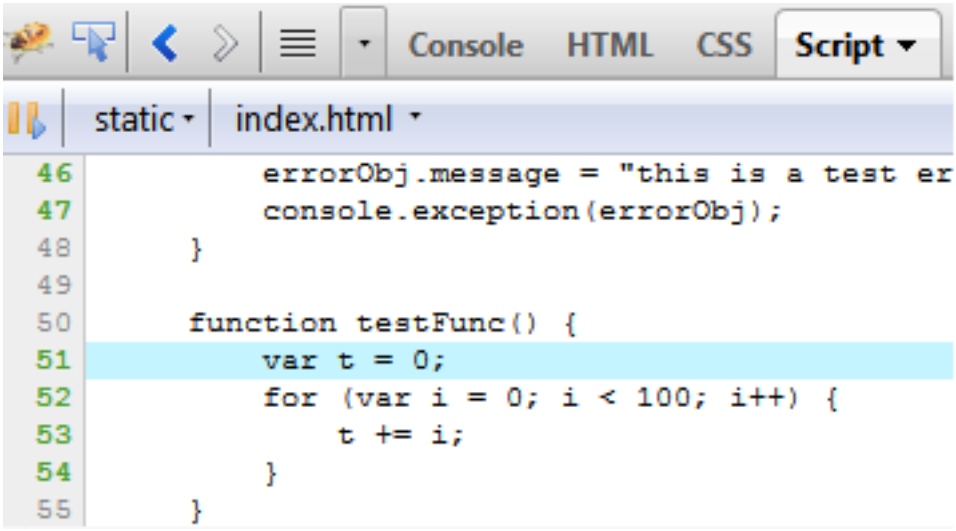
عملکرد 3 جایگزین نخست با توجه با مثال قبل مشخص شد . پس به سراغ جایگزین o% و c% می‌رویم .
اگر در رشته‌ی مورد نظر ، یک شیء (تابع ، آرایه ، ...) برای جایگزین o% ارسال کنیم ، در خروجی آن شیء بصورت لینک نمایش داده می‌شود که با کلیک بروی آن ، فایرباگ آن شیء را در تب مناسبش Inspect می‌کند .
مثال :

```
console.log("this is a test functin : %o",testFunc);
```

نتیجه :

```
>>> console.log("this is a test functin : %o",testFunc);
this is a test functin : testFunc()
```

و زمانی که بروی لینک testFunc کلیک کنیم :



یک ترفند : بوسیله جایگزین o% توانستیم به تابع مورد نظر لینک بدهیم . اگر بجای جایگزین o% از s% استفاده کنیم ، می‌توانیم بدنه‌ی تابع را ببینیم :

```
console.log("this is a test functin : %s",testFunc);
```

نتیجه :

```
>>> console.log("this is a test functin : %s",testFunc);
this is a test functin : function testFunc() {
    var t = 0;
    for (var i = 0; i < 100; i++) {
        t += i;
    }
}
```

توسط جایگزین %c هم می‌توانید خروجی را فرمت کنید .

```
console.log("%cThis is a Style Formatted Log","color:green;text-decoration:underline;");
```

نتیجه :

```
>>> console.log("%cThis is a Style Formatted Log","color:green;text-decoration:underline;");
This is a Style Formatted Log
```

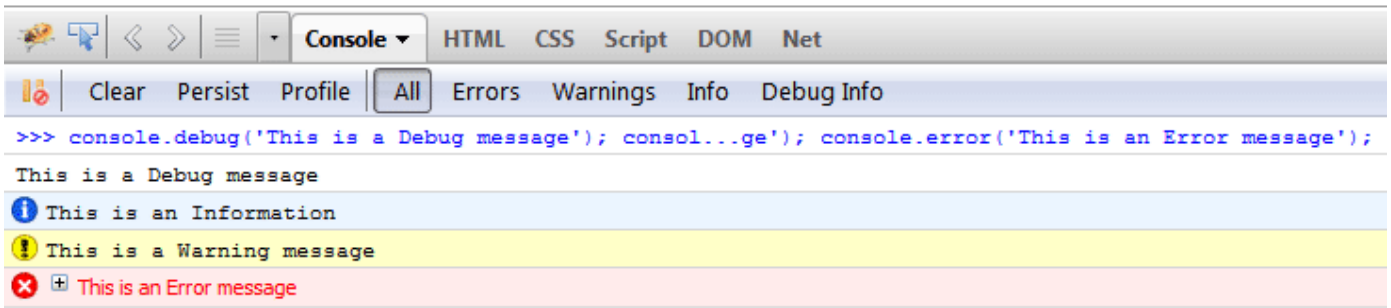
([... ,console.debug(object[, object

([... ,console.info(object[, object

([... ,console.warn(object[, object

([... ,console.error(object[, object

مشابه با دستور log عمل می‌کنند با این تفاوت که خروجی را با استایل متفاوتی نمایش می‌دهند .
همچنین هر یک از این دستورات ، توسط دکمه‌های همنام در کنسول قابل فیلتر شدن هستند .

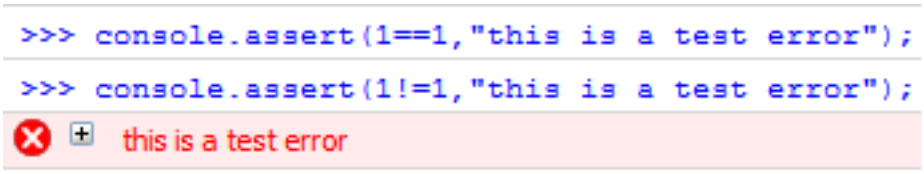


([... ,console.assert(expression[, object

چک می‌کند که عبارت ارسال شده true هست یا نه . اگر true نبود ، پیغام وارد شده را چاپ و یک استثناء ایجاد می‌کند .

```
console.assert(1==1,"this is a test error");
console.assert(1!=1,"this is a test error");
```

نتیجه :



()console.clear

(console.dir(object

(console.dirxml(node

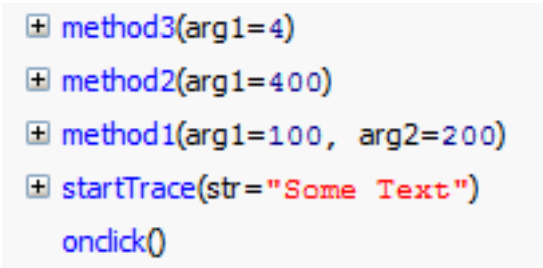
([console.profile([title

()console.profileEnd

این توابع معادل توابع همنامشان در خط فرمان هستند که در قسمت قبل با عملکردشان آشنا شدیم .

()console.trace

با این متد می‌توانید پی ببرید که از کجا و توسط چه متدهایی برنامه به قسمت trace رسیده . برای درک بهتر مجددا اسکریپت صفحه‌ی تست این مقاله را بررسی کنید (جایی که متد trace قرار داده شده است) .
اکنون صفحه‌ی تست را باز کنید و بروی دکمه‌ی startTrace کلیک کنید . خروجی ظاهر شده در کنسول را از پایین به بالا بررسی کنید .



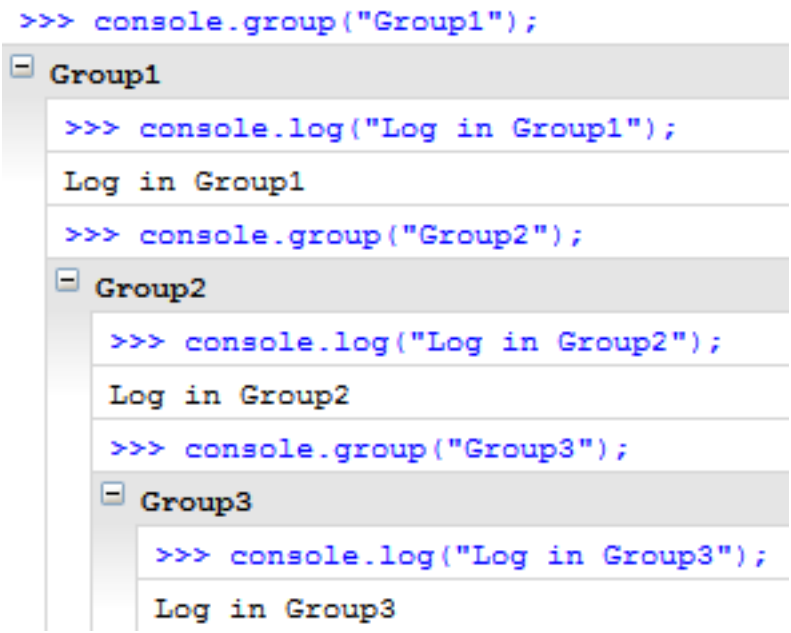
حتما متوجه شدید که متد method3 چگونه در کدهایمان فراخوانی شده است !؟
ابتدا با کلیک بروی دکمه‌ی startTrace ، متد startTrace اجرا شده و به همین ترتیب متد startTrace متد method1 ، متد method1 هم متد method2 و در نهایت method2 متد method3 را فراخوانی کرده است .
دستور trace زمانی که در حال بررسی کدهای برنامه نویسان دیگر هستید ، بسیار می‌تواند به شما کمک کند .

([... ,console.group(object[, object

با این دستور می‌توانید لاگ‌های کنسول را بصورت تو در تو گروه بندی کنید .

```
console.group("Group1");
console.log("Log in Group1");
console.group("Group2");
console.log("Log in Group2");
console.group("Group3");
console.log("Log in Group3");
```

نتیجه :



([... ,console.groupCollapsed(object[, object

این دستور معادل دستور قبلی است با این تفاوت که هنگام ایجاد ، گروه را جمع می‌کند .

()console.groupEnd

به آخرین گروه بندی ایجاد شده خاتمه می‌دهد .

(console.time(name

یک تایمر با نام داده شده ایجاد می‌کند . زمانی که نیاز دارید زمان طی شده بین 2 نقطه را اندازه گیری کنید ، این تابع مفید خواهد بود .

(console.timeEnd(name

تایمر همنام را متوقف و زمان طی شده را چاپ می‌کند .

```
console.time("TestTime");
var t = 1;
for (var i = 0; i < 100000; i++) { t *= (i + t) }
console.timeEnd("TestTime");
```

نتیجه :

```
>>> console.time("TestTime");  var t = 1;  for (var ...) { t *= (i + t) }
console.timeEnd("TestTime");
TestTime: 527ms
527
```

console.timeStamp()

توضیحات کامل را از اینجا دریافت کنید .

console.count([title])

تعداد دفعات فراخوانی شدن کدی که این متد در آنجا قرار دارد را چاپ می‌کند .
البته ظاهرا در ورژن 10.0.1 که بنده با آن کار می‌کنم ، این دستور بی عیب کار نمی‌کند . زیرا بجای آنکه در هربار فراخوانی ، در همان خط تعداد فراخوانی را نمایش بدهد ، فقط اولین لاگ را آپدیت می‌کند .

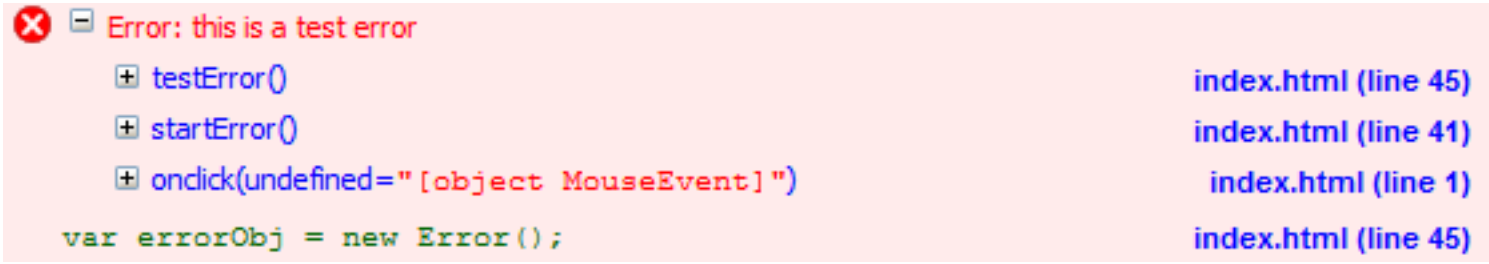
```
>>> testCount()
testCount() Calls Count . 5
undefined
>>> testCount()
undefined
>>> testCount()
undefined
>>> testCount()
undefined
>>> testCount()
undefined
```

console.exception(error-object[, object, ...])

یک پیغام خطا را به همراه ردیابی کامل اجرای کدها تا زمان رویداد خطا (مانند متد trace) چاپ می‌کند .
در صفحه‌ی تست این متد را اجرا کنید :

```
startError();
```

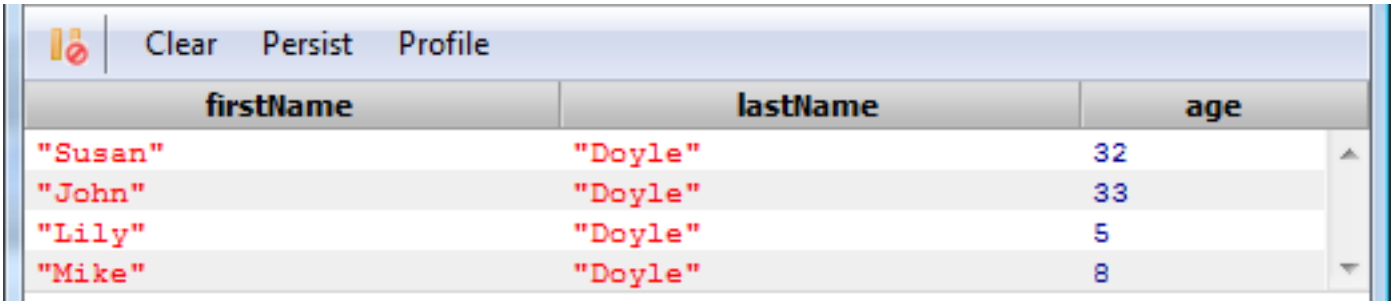
نتیجه :



توجه کنید که ما برای مشاهده‌ی عملکرد صحیح این دستور ، آن را در تابع testError قرار دادیم و بوسیله تابع startError آن فراخوانی کردیم .

console.table(data[, columns])

بوسیله این دستور می‌توانید مجموعه ای از اطلاعات را بصورت جدول بندی نمایش بدهید .
این متد از متدهای جدیدی است که در فایرباگ قرار داده شده است .



برای اطلاعات بیشتر به اینجا مراجعه کنید .

منابع : https://getfirebug.com/wiki/index.php/Console_API

نظرات خوانندگان

نویسنده: مرتضی
تاریخ: ۱۳۹۱/۰۵/۱۱ ۱۳:۷

با سلام
می خواستم بپرسم فایرباگ چه محاسنی نسبت به developer tool مرورگر کروم داره؟ به نظر خیلی شبیه به هم می‌آیند.
با تشکر از زحمات شما

نویسنده: احمد احمدی
تاریخ: ۱۳۹۱/۰۵/۱۲ ۷:۱۲

سلام
بنده خیلی کم با Developer Tool کار کردم . اما به نظر میرسه قوی و کامل‌تر از فایرباگ باشه .
نتایج جستجوی عبارت " google chrome developer tools vs firebug " در گوگل هم جالب و مفیده .