

خطایابی در کروم (debugging)

@alithecodeguy

قبل از اینکه کدهای پیچیده تری بنویسیم ، کمی در مورد خطایابی (دیباگ) صحبت کنیم.

دیباگ کردن به فرآیند یافتن خطاها و برطرف سازی آنها در سورس کد می گویند . اکثر مرورگرهای مدرن و بیشتر محیطهای توسعه از ابزارهای دیباگ کردن پشتیبانی می کنند . ابزار دیباگ ، ابزاری است که فرآیند دیباگ کردن را بسیار آسان می کند . این ابزارها ، همچنین اجازه اجرای خط به خط کد و بررسی نحوه اجرای کد را در اختیار توسعه دهنده قرار می دهند .

در این آموزش از مرورگر کروم استفاده خواهیم کرد چرا که ویژگی های کافی را داشته و همچنین با ابزارهای سایر مرورگرها تفاوت آنچنانی ندارد .

@alithecodeguy

پنل Sources

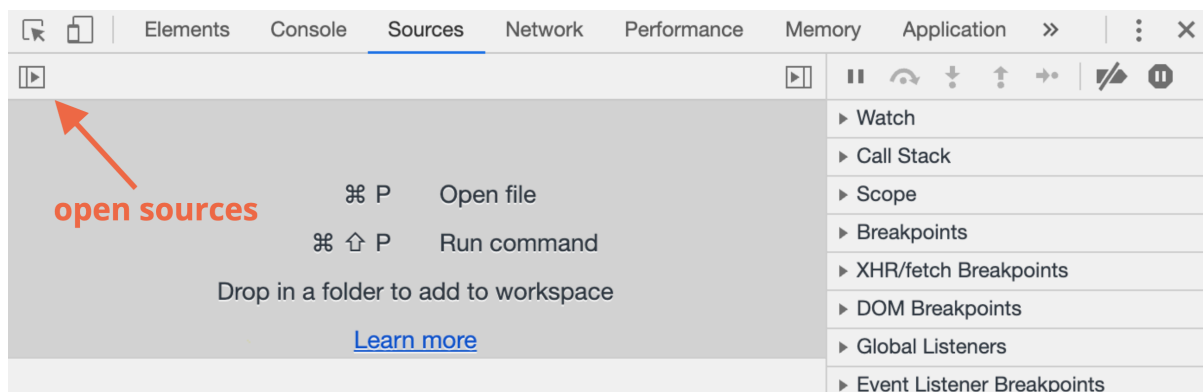
ظاهر مرورگر کرومی که شما استفاده می کنید ممکن است با تصاویری که در ادامه آمده است کمی متفاوت باشد ولی تفاوتشان آنقدری نخواهد بود که دچار مشکل شوید .

- لینک زیر را در کروم باز کنید :

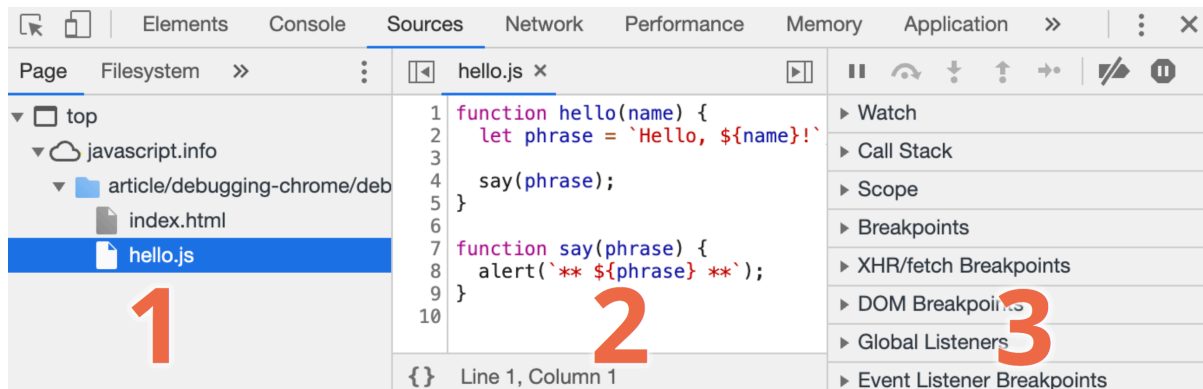
<https://javascript.info/article/debugging-chrome/debugging/index.html>

- برای باز کردن developer tools در ویندوز از کلید F12 و در مک از کلید ترکیبی Cmd+Opt+I استفاده نمایید .

- تب Sources را انتخاب نمایید و روی علامت open sources کلیک کنید .



فایل hello.js را از منوی سمت چپ انتخاب نمایید . چیزی که مشاهده خواهید کرد مشابه تصویر زیر خواهد بود :



پنل Sources سه بخش دارد :

1. File Navigator : لیست تمامی فایل ها در این قسمت ظاهر می شود. افزونه های کروم نیز در این قسمت ظاهر

می شوند.

2. Code Editor : سورس کد را نمایش می دهد.

3. JavaScript Debugging : برای دیباگ کردن به کار می رود و در ادامه آن را بررسی خواهیم نمود.

Console

اگر کلید Esc را بفشارید ، console در پایین صفحه باز خواهد شد . هر دستوری را می توانید داخل آن نوشته و برای اجرای آن کلید Enter را بفشارید .

بعد از اجرای دستور ، نتیجه آن نمایش داده خواهد شد .

برای مثال ، نتیجه 1+2 ، 3 خواهد شد و عبارت hello("hello") کاری انجام نمی دهد پس نتیجه آن undefined خواهد شد .



Breakpoints

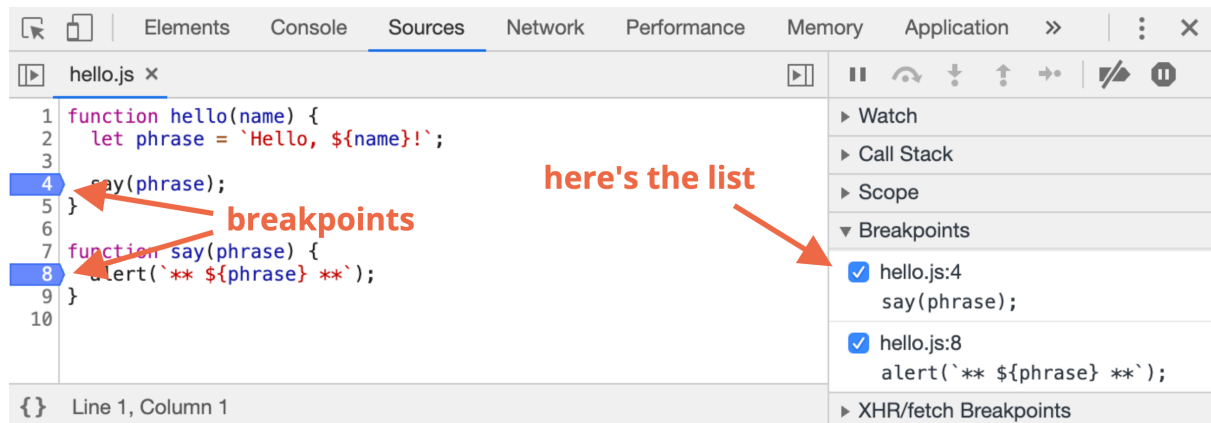
بباید بررسی کنیم که در سورس کد لینک فوق ، چه اتفاقی رخ داده است . در سورس hello.js مستقیماً روی عدد 4 کلیک کنید .

تبریک! اولین breakpoint خود را استفاده کردید!

روی عدد 8 در خط 8 نیز کلیک کنید .

@alithecodeguy

پنل شما مشابه تصویر زیر خواهد شد :



breakpoint نقطه ای از کد است که debugger در آن نقطه ، اجرای کد را به صورت موقت ، متوقف خواهد کرد. تا زمانی که کد متوقف شده است ، قادر خواهید بود که متغیرهای جاری را بررسی کرده و دستورات مختلفی را اجرا نمایید. به عبارت دیگر قادر خواهید بود آن را خطایابی (دیباگ) کنیم. لیست breakpoint ها در سمت راست نمایش داده می شود. اگر تعداد زیادی breakpoint در تعداد زیادی فایل داشته باشید ، پنل سمت راست بسیار کارا خواهد بود. این پنل به ما امکان این را می دهد که :

- مستقیماً سراغ breakpoint مد نظرمان برویم.
- به صورت موقت breakpoint خاصی را غیر فعال کنیم.
- با کلیک روی breakpoint و انتخاب گزینه remove ، breakpoint مورد نظر را حذف کنیم.

کلیک راست روی اعداد منجر به ایجاد breakpoint های شرطی می شود که فقط به شرطی اجرا خواهند شد که مقدار عبارت وارد شده برای آن truthy شود. استفاده از این روش هنگامی که می خواهید فرآیند اجرای کد را به ازای مقدار خاصی (یا موارد مشابه) متوقف کنید ، مفید خواهد بود.

دستور Debugger

برای توقف اجرای برنامه از دستور debugger نیز می توانید استفاده کنید :

```
function hello(name) {  
  let phrase = `Hello, ${name}!`;  
  debugger; // <-- the debugger stops here  
  say(phrase);  
}
```

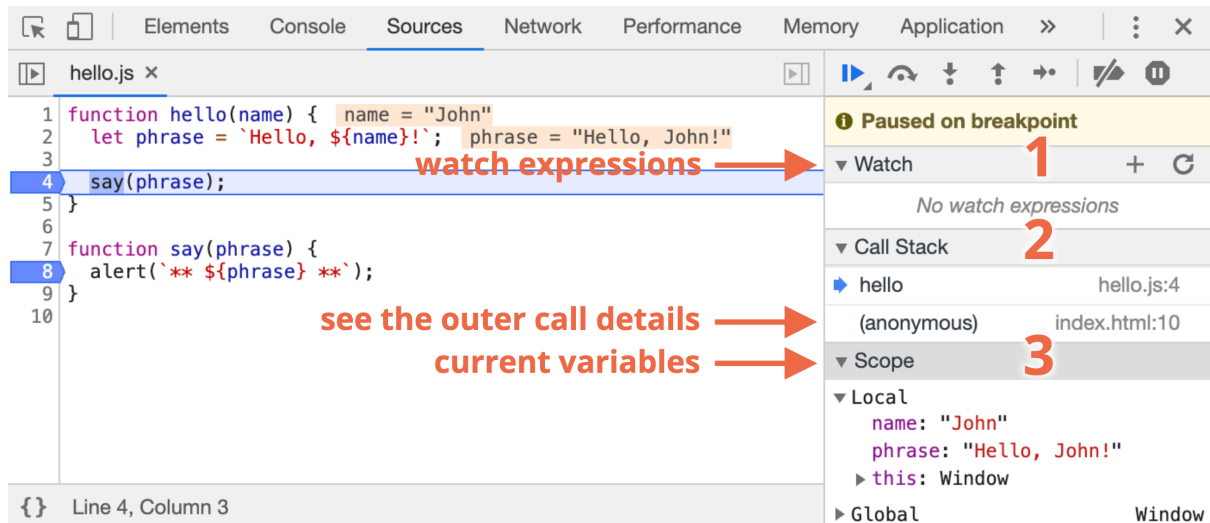
@alithecodeguy

هنگامی که در محیط ویرایشگر در حال کدزنی هستیم و نمی خواهیم که به مرورگر مراجعه کنیم ، از این روش استفاده می کنیم.

بررسی پنل در حین توقف اجرای کد

در مثال بالا ، فانکشن hello() در حین بارگذاری صفحه فراخوانی می شود پس راحت ترین راه برای فعال کردن debugger ، رفرش صفحه است. برای اینکار از کلید F5 در ویندوز و از کلید ترکیبی Cmd+R در مک استفاده نمایید.

اگر breakpoint به درستی استفاده شده باشد ، اجرای کد در خط 4 متوقف می‌شود:



باکس های که با فلش قرمز نشان داده شده است را باز نمایید . این باکس ها ، اجازه بررسی وضعیت جاری کد را در اختیار قرار می‌دهد :

1. Watch : مقدار جاری هر عبارتی را نمایش می‌دهد .

می‌توانید بر روی علامت + کلیک کرده و عبارتی را وارد نمایید . debugger مقدار آن را در هر لحظه از اجرای کد نمایش داده و در طول فرآیند اجرا ، مقدار آن را به صورت خودکار بروزرسانی می‌کند .

2. Call Stack : فراخوانی های تودرتو را نمایش می‌دهد .

در لحظه جاری ، debugger درون فانکشن hello() قرار دارد که توسط اسکریپتی که داخل index.html است فراخوانی شده است . چون فانکشن دیگری آن را فراخوانی نکرده است ، از کلمه anonymous استفاده کرده است . اگر روی آیتم خاصی درون stack کلیک کنید ، debugger به سراغ آن می‌رود .

3. Scope : متغیرهای جاری را نمایش می‌دهد .

1. local : متغیرهای محلی فانکشن ها را نمایش می‌دهد . همچنین خواهید دید که مقادیر آنها درون سورس کد هایلايت شده‌اند .

2. Global : متغیرهای global را نمایش می‌دهد . (متغیرهایی که خارج از هر فانکشنی هستند .)

trace کردن روند اجرا (خط به خط دنبال کردن روند اجرا)

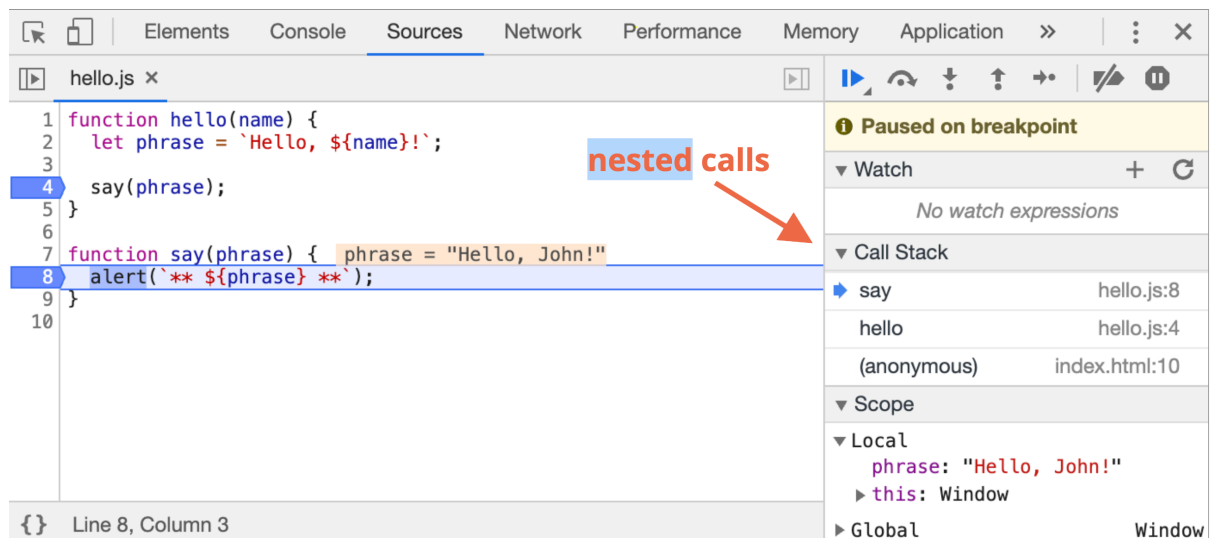
حال زمان trace کردن اجرای اسکریپت فرا رسیده است . برای این منظور دکمه های بالا و سمت راست پنل وجود دارد که آنها را بررسی خواهیم کرد :

@alithecodeguy

• Resume ►► : اجرا را ادامه می‌دهد و کلید میانبر آن F8 است .

فرآیند اجرا را ادامه می‌دهد . اگر breakpoint دیگری نباشد ، اجرا را ادامه داده و دیباگر کنترل اجرا را از دست می‌دهد .

بعد از کلیک روی آن با صحنه‌ای مشابه تصویر زیر روبرو خواهید شد :



اجرا ادامه پیدا کرده و به breakpoint دیگری درون `say()` رسیده و متوقف می‌شود. نگاهی به CallStack در سمت بیندازید. یک فراخوانی دیگر به آن اضافه گردید. اکنون دیباگر داخل فانکشن `say()` قرار دارد.

• **Step** : دستور بعدی را اجرا می‌کند و کلید میانبر آن F9 است.

دستور بعدی را اجرا می‌کند. اگر اکنون روی آن کلیک کنید، `alert` اجرا خواهد شد. کلیک‌های متعدد روی این دکمه باعث اجرای خط به خط کل اسکریپت خواهد شد.

• **Step over** : دستور بعدی را اجرا می‌کند ولی وارد فانکشن نمی‌شود و کلید میانبر آن F10 است.

این کلید رفتاری مشابه کلید قبلی دارد ولی اگر دستور بعدی `function` باشد رفتار متفاوتی نمایش می‌دهد. منظورمان از فانکشن‌های داخلی مانند `alert` نمی‌باشد بلکه فانکشن‌هایی است که توسط برنامه نویس ایجاد شده است.

کلید **Step** وارد فانکشن شده و اجرا را در خط اول آن فانکشن متوقف می‌کند ولی **StepOver** اجرای فانکشن را به صورت پنهان انجام داده و دیباگر را وارد فانکشن نمی‌کند و دیباگر را بلافاصله پس از اجرای آن فانکشن متوقف می‌کند. در صورتی که تمایل نداشته باشیم که بدانیم درون فانکشن چه اتفاقی می‌افتد، از این دکمه استفاده می‌کنیم.

• **Step Into** : کلید میانبر آن F11 است.

این کلید رفتاری مشابه **Step** دارد به جز اینکه در مورد فانکشن‌های `async` متفاوت رفتار می‌کند. اگر جاوا اسکریپت را به تازگی شروع کرده‌اید می‌توانید این گزینه را نادیده بگیرید چرا که فعلاً فانکشن `async` ای را فراخوانی نکرده‌ایم.

فقط توجه داشته باشید در حالی که کلید **Step** فراخوانی فانکشن‌های `async` را نادیده می‌گیرد (مانند `setTimeout()`) ولی کلید **Step into** درون آنها رفته و در صورت لزوم منتظر می‌ماند.

• **Step out** : اجرا را تا پایان فانکشن جاری ادامه می‌دهد و کلید میانبر آن Shift+F11 است.

فرآیند اجرا را ادامه داده و در آخرین خط فانکشن جاری متوقف می‌شود. هنگامی که روی کلید **Step** اشتباهی کلیک کرده ایم و وارد فانکشن شده‌ایم، می‌توانیم از این کلید استفاده کرده و از آن خارج شویم.

• **enable/disable all breakpoints** : این کلید با فرآیند اجرا کاری ندارد. فقط breakpoint ها را فعال و یا غیر فعال می‌کند.

• **enable/disable automatic pause in case of an error** : در صورتی که این کلید فعال باشد و

developer tools نیز باز باشد، خطای موجود در اسکریپت، اجرا را متوقف می‌کند، سپس می‌توانیم متغیرها را بررسی کنیم

و ببینیم چه چیزی باعث بروز خطا شده است. بنابراین اگر خطایی منجر به پایان یافتن اسکریپت می شود ، با فعال کردن این گزینه و رفرش کردن صفحه می توان آن را پیدا کرد.

اگر روی هر یک از خطهای کد کلیک راست کنید گزینه ای با عنوان continue to here در اختیار شما قرار می گیرد.

این گزینه هنگامی که بخواهیم چند مرحله جلوتر برویم ولی آنقدر تنبل هستیم که نمی خواهیم از breakpoint استفاده کنیم.

@alithecodeguy

Logging

برای اینکه چیزی را در کنسول نمایش دهیم می توانیم از فانکشن console.log استفاده نماییم.

برای مثال اگر بخواهیم مقادیر 1 الی 4 را در کنسول نمایش دهیم می توانیم از کد زیر استفاده کنیم :

```
for (let i = 0; i < 5; i++) {  
  console.log("value", i);  
}
```


کاربران عادی این خروجی را نخواهد دید چرا که در کنسول نمایش داده می شود. برای مشاهده این خروجی ، در ابزار developer tools وارد پنل console شده و یا هنگامی که در developer tools هستید کلید Esc را بفشارید .
اگر از ابزار logging درست استفاده کنیم ، برای بررسی و رفع خطای اسکریپت ، نیاز به debugger نخواهیم داشت.

خلاصه

همانطور که دیدیم سه راه کلی برای توقف اجرای یک اسکریپت به صورت موقت وجود دارد :

1. breakpoint

2. عبارت debugger

3. بروز خطا و فعال سازی دکمه 

@alithecodeguy

هنگامی که کد متوقف شده است ، می توانیم متغیرهای آن را بررسی کرده و کد را trace کنیم تا متوجه شویم چه چیز باعث بروز خطا شده است .

developer tools گزینه های فراوانی دارد که فقط تعداد اندکی از آنها را بررسی کردیم .

اطلاعاتی که در این بخش ارائه شد ، برای شروع debugging کفایت می کند ولی در صورتی که قصد دارید از امکانات پیشرفته آن استفاده کنید ، پیشنهاد می شود که به لینک زیر مراجعه کرده و مستندات developer tools را مطالعه کنید .

<https://developers.google.com/web/tools/chrome-devtools>.

اوه! البته می توانید آزادانه به هر قسمت آن سر بزنید و ببینید که چه اتفاقی می افتد . کلیک راست را فراموش نکنید!