Stupid JavaScript debugging tricks: Abusing the conditional breakpoint

devblogs.microsoft.com/oldnewthing/20151119-00

November 19, 2015

Raymond Chen

Your favorite JavaScript debugger may very well have a conditional breakpoint facility, and if it does, you can abuse it to do things unrelated to conditional breakpoints.

Since conditional breakpoints are based on evaluating an expression, you can use an expression with side effects, like, say, logging.

Breakpoint Condition

When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is true or has changed.

☒ Condition:

console.log(“click received”)

⊙ Is true

○ Has changed

OK Cancel

The `console.log` function returns `undefined`, so the condition is never true, but that’s okay, because we’re executing the expression for its side effect of printing a string to the console.

Here, we just logged a hard-coded string, but you can log any expression you want. For example, you could use

```
console.log("current selection is " + this.currentSelection)
```

Or even more fun:

```
console.log(getStack())
```

where you've defined

```
function getStack() { try { throw new Error(); } catch (e) { return e.stack; } }
```

What, you didn't define that function? No problem. Just evaluate it in your immediate window and boom, now it's defined.

Now, printing an entire stack trace may be excessive. So store it in your object for future inspection. For example, you might do this in a constructor so that you have a stack trace of how your object got created. That way, when you have a transaction that failed to complete, you can look at the cached stack trace to see how the transaction got started.

```
(this._stack = getStack()) === 42
```

The extra `=== 42` at the end is to ensure that the value of the expression is falsy.

Anyway, just a few quick tips for JavaScript debugging. I didn't come up with these ideas, but I'm sharing them.

Raymond Chen

Follow