Chapter 98: Global error handling in browsers

Parameter	Details
eventOrMessage	Some browsers will call the event handler with just one argument, an `Event` object. However, other browsers, especially the older ones and older mobile ones will supply a `String` message as a first argument.
url	If a handler is called with more than 1 argument, the second argument usually is an URL of a JavaScript file that is the source of the problem.
lineNumber	If a handler is called with more than 1 argument, the third argument is a line number inside the JavaScript source file.
colNumber	If a handler is called with more than 1 argument, the fourth argument is the column number inside the JavaScript source file.
error	If a handler is called with more than 1 argument, the fifth argument is sometimes an `Error` object describing the problem.

Section 98.1: Handling window.onerror to report all errors back to the server-side

The following example listens to `window.onerror` event and uses an image beacon technique to send the information through the GET parameters of an URL.

```
var hasLoggedOnce = false;

// Some browsers (at least Firefox) don't report line and column numbers
// when event is handled through window.addEventListener('error', fn). That's why
// a more reliable approach is to set an event listener via direct assignment.
window.onerror = function (eventOrMessage, url, lineNumber, colNumber, error) {
    if (hasLoggedOnce || !eventOrMessage) {
        // It does not make sense to report an error if:
        // 1. another one has already been reported -- the page has an invalid state and may produce
way too many errors.
        // 2. the provided information does not make sense (!eventOrMessage -- the browser didn't
supply information for some reason.)
        return;
    }
    hasLoggedOnce = true;
    if (typeof eventOrMessage !== 'string') {
        error = eventOrMessage.error;
        url = eventOrMessage.filename || eventOrMessage.fileName;
        lineNumber = eventOrMessage.lineno || eventOrMessage.lineNumber;
        colNumber = eventOrMessage.colno || eventOrMessage.columnNumber;
        eventOrMessage = eventOrMessage.message || eventOrMessage.name || error.message ||
error.name;
    }
    if (error && error.stack) {
        eventOrMessage = [eventOrMessage, '; Stack: ', error.stack, '.'].join('');
    }
    var jsFile = (/[^/]+\.js/i.exec(url || '') || [])[0] || 'inlineScriptOrDynamicEvalCode',
        stack = [eventOrMessage, ' Occurred in ', jsFile, ':', lineNumber || '?', ':', colNumber ||
'?'].join('');

    // shortening the message a bit so that it is more likely to fit into browser's URL length limit
(which is 2,083 in some browsers)
    stack = stack.replace(/https?\:\/\/[^/]+/gi, '');
    // calling the server-side handler which should probably register the error in a database or a
log file
```

```
    new Image().src = '/exampleErrorReporting?stack=' + encodeURIComponent(stack);

    // window.DEBUG_ENVIRONMENT a configurable property that may be set to true somewhere else for
debugging and testing purposes.
    if (window.DEBUG_ENVIRONMENT) {
        alert('Client-side script failed: ' + stack);
    }
}
```