Camlin Page
Date / /

Error handling in Javascript

Writing programs that work when everything goes ~~goe~~ expected is a good start. Making your programs behave properly when encountering unexpected conditions is where it really gets challenging. So, to make sure our program will work within any condition without fail we should follow certain rules ~~and~~ or procedures.

Rule #1:-

Assume your code will fail:-

After writing a code we should have a Assuptions that what if a variable is Null and what. For example:

```
Object.expand = function(destination, source) {
    for (var property in soure)
        if destination[property] = source[property];
        return destination;
    };
```

Here in the above code, what if the user doesnot gives the destination ~~or~~ or source code?

Rule #2:- Log the mistakes (errors) to server:-

We should always try to log the errors to the server. If ~~we~~, incase we are logging the errors to the user screen. It would address to Vulnarable attack.

Simple logging

```
function log(sev, msg) {
    var img = new Image();
    img.src = "log.php?sev=" + encodeURIComponent(sev)
              + encodeURIComponent(msg);
}
log('3', "something happened");
```

Camlin Page
Date / /

Rule3:-

try-catch

The try...catch statement marks a block of statements to try and specifies a response should an exception be thrown.

For example:-

```
try{
    operation();
}catch(ex){
    log(2, "operation() failed: "+exc.message);
}
```

Window.onerror

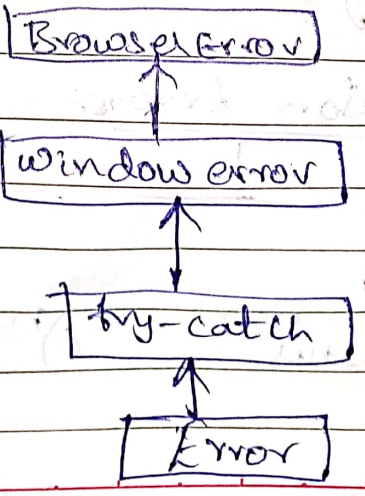
Window.onerror is a DOM event handler that acts like a global try...catch. This is great for catching unexpected expectations.

For example:-

```
Window.onerror = function(msg, url, line){
    log(1, msg);
    return true;
};
```

but it is only supported in IE and firefox.

Error-Lifecycle:-



Camlin Page
Date / /

Rule 4:- Identify where the errors might occur!-

3 types of Errors:-

1 - type coercion errors
2 - Data type errors
3 - Communication errors.

1) type coercion errors

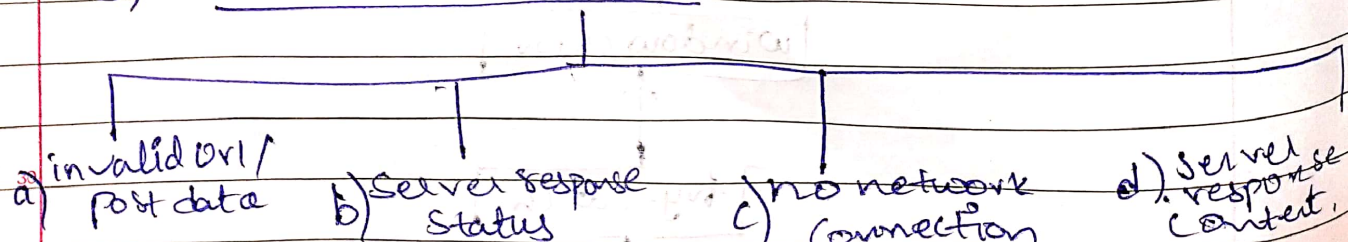
```
//Boolean conversion
if (null) {/* never executed */}
if (0) {/* never executed */}
if ("") {/* never executed */}

if ("hi") {/* executed */}
if (-1) {//executed}
if ([]) {//executed}.
//equivalence conversion
alert (5 == "5"); //true
alert (5 === "5"); //false
alert (1 == true); //true
alert (1 === true); //false
```

2) Datatype errors:-

- often occurs with function arguements
- Typically a symtom of insufficient value checking.

3) Communication errors:-



Camlin Page
Date / /

a) Invalid URL/postdata
- Typically long string concatenations.
- Don't forget to use encodeURLComponent() on each part.
- make sure parameters are named correctly.

b) Server Response status:-
- 200 is not only valid status that may be returned, because of 304.
- Any other status means you didn't get data.

c) No Network connection:-
- Internet explorer throws an error when calling open() but then goes to normal lifecycle.
- Firefox fails silently but throws an error if you try to access.

d) Server Response content:-
- A server of 200/304 is not enough
- Server errors often return HTML.
- If possible, set status to 500.

Rule 5:- Throw your own errors.-
Throw (or) try catch
- Errors should be thrown in the low-level parts of the application.
- use try-catch blocks in higher level parts.

Camlin Page
Date / /

Rule 6 : Distainguish fatal versus non-fatal

Non-fatal Errors

- Won't interfere with user's main tasks.
- Affects only a portion of the page
- ~~Readable~~ Recovery is possible
- A repeat of action may result in the appropriate result.

Fatal Errors

- The application absolutely cannot continue.
- Significantly interferes with users ability to be productive
- Other errors will occur if the application continues.
- Message the user immediately.
- Reload.

Rule 7:- Provide a debug mode

- Assign a variable that is globally available.
- try catch should re-throw the error.
- window.onerror should return false,
- Allow the browser to handle error.