JavaScript Error Handling

Let me start with a quote of **Donald A. Norman ~**

> *"If an error is possible, someone will make it. The designer must assume that all possible errors will occur and design so as to minimize the chance of the error in the first place, or its effects once it gets made. Errors should be easy to detect, they should have minimal consequences, and, if possible, their effects should be reversible."*

Note: Link to the task slide is at the bottom.

Rule 1 ⇒ Assume your code will fail.

- It can happen from the destination or source.
- Maybe the destination is empty or the source is empty.
- The destination may not have met the requirements of the source.
- Dependency issues or whatever… There is a chance for code to fail.

Rule 2 ⇒ Log errors to the server.

- Logging the errors to the server makes the code more reliable.
- Logging helps us to understand if the code is working properly.
- It helps us the developers to understand where the code is getting trouble or error.
- This helps us to write better code from failures.

Rule 3 ⇒ You, not the browser, handle errors.

- As a developer, we have to handle the errors happening on the client-side.
- Every code is prone to errors, so we have to identify where the error might happen and we have to handle the errors properly.
- The basic example of error handlers is *try-catch.*
- And before the error execution in the browser, there is an another method called window.onerror.
- That is the last stop before the browser responds.
- If the onerror is true, the browser will not respond to the error.
- The flow chart is like:

- Error → try-catch → window.onerror → Browser error

Rule 4 ⇒ Identify where the error might occur.

Type of errors

- **Type coercion errors**
  - Type coercion is the process of converting a value from one type to another.
  - Type coercion error happens when there is a problem in the type coercion.
- **Data type errors**
  - Often occurs with the function arguments.
  - Typically a symptom of insufficient value checking.
  - The error occurs when the expected type of data is not equal to the actual type of data.
- **Communication errors**
  - **Invalid URL / post data**
    - Typically long string concatenations.
    - Essential to use encodeURIComponent() in each instance of certain characters by one, two, three, or four escape sequences representing the UTF-8 encoding of the character.
    - Not - encodeURI()
    - Need to make sure the parameters are named correctly.
  - **Server response status**
    - 200 is not the only valid status that may be returned.
    - We have to be aware of error 304.
    - If we get any other status → we didn't get the data.
  - **No network connection**
    - Internet explorer throws an error when calling open() but then goes through the normal lifecycle.
    - Firefox fails silently but throws an error if you try to access any response property (status, statusText, responseText).
  - **Server response content**
    - A status of 200 / 304 is not enough.
    - Server errors often return HTML.
    - If possible, set the status to 500.

Rule 5 ⇒ Throw your own errors.

We can throw our own errors in the browser. That is how most of the errors occur in the browser. Throwing error is an advantage we have as a developer. The error helps users to know where the real problem is to view a webpage or web application. When we know where the error might occur, we throw an error with if condition or try-catch method. Now that error we throw will be responded or addressed by the browser.

- Errors should be thrown in the low-level parts of the application.
    - utilities, core libraries, etc…
- Use try-catch blocks at higher level parts.
    - Application-specific
    - Client-side business logic

Rule 6 ⇒ Distinguish fatal and non-fatal.

Fatal errors

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

Non-Fatal errors

- Won't interfere with the user's main tasks.
- Affects only a portion of the page.
    - Easily disabled or ignored
- Recovery is possible.
- A repeat of the action may result in an appropriate result.
- Don't tell the user it isn't working unless absolutely necessary.

Fatal or Non-Fatal?

- Don't allow your code to determine what is and is not fatal.
    - Watch out for loops.
- The user's experience comes first.

Rule 7 ⇒ Provide a debug mode.

Debug mode

- Assign a variable that is globally available.
- Try-catch should re-throw the error. (Using try-catch inside for-loop to throw the error again and again)
- Window.one error should return false. (window.one should be false if debug mode is ‘on’)
- Allow the browser to handle the error. (Let the browser handle some errors)

Link to the task slide :
https://www.slideshare.net/nzakas/enterprise-javascript-error-handling-presentation/45-S_ummary

---

Anandha Narayanan B
BATCH: B15 WE