Edgemony

Coding Bootcamp

JavaScript: Drive in

Error Handling

Gestione e tipologie di errori

In JavaScript ci sono due macro categorie di errori:

- **Parse Errors.** Errori che rendono il codice non leggibile dalla macchina perché la sintassi non rispetta le regole del linguaggio.
- **Runtime Errors.** Errori generati durante l'esecuzione del codice.

Solamente il secondo tipi di errori è gestibile all'interno del codice.

PARSE ERRORS

```
{{ let myVar = 2 // Questo tipo di errore rende il codice illeggibile e quindi non eseguibile
```

RUNTIME ERRORS

```
console.log(myUndeclaredVar) // Questo errore è sintatticamente corretto, il codice verrà eseguito ma
si bloccherà in questo punto perché la variabile myUndeclaredVar non è mai stata dichiarata.
```

try…catch

Il costrutto `try…catch` permette di «catturare» e gestire errori *runtime* evitando che l'esecuzione del codice si interrompa:

```
try {
  console.log("Questa codice viene eseguito 🤓")
  myUndeclaredVar // questa variabile non è mai stata dichiarata, quindi viene generato un errore
  console.log("Questo purtroppo no 😢, il codice salta direttamente all'istruzione «catch»")
} catch (err) {
  console.error(err) // ReferenceError: muUndeclaredVar is not defined
}

alert("Woww Il mio codice non si è interrotto! 😍")
```

MDN try…catch link

L'oggetto Error

JavaScript fornisce una serie standard di errori built-in, che vengono generati durante l'esecuzione del codice nel caso vengano riscontrati degli errori.

L'oggetto padre di tutti questi errori è `Error` che possiede due proprietà standard leggibili all'interno dell'istruzione `catch: name e message.`

```
try {
  blablabla;
} catch (err) {
  console.log(err.name) // ReferenceError, SynthaxError, TypeError, ecc.
  console.log(err.message) // myUndeclaredVariable is not defined
}
```

Oltre ad `Error` JavaScript fornisce altri errori derivati: `EvalError, RangeError, ReferenceError, SynthaxError, TypeError, URIError, AggregateError, InternalError.`

MDN Oggetto Error

L'operatore «throw»

Grazie all'operatore `throw` è possibile generare degli errori in maniera programmatica.
Questi errori possono poi essere «catturati» e gestiti all'interno del blocco `catch`:

```
try { // Evitare
  throw "Ecco un errore";
} catch (err) {
  console.error(err) // Ecco un errore
}

try { // Corretto
  throw new Error("Ecco un errore")
} catch (err) {
  console.error(err) // Error: Ecco un errore
}
```

Tecnicamente è possibile generare qualsiasi tipo di errore con l'operatore `throw` (stinghe, numeri, booleani, oggetti, ecc.) ma è buona prassi utilizzare l'oggetto `Error` (o i suoi derivati), personalizzando il `message`.

MDN operatore throw

«rethrow» di errori

Quando vengono generati errori programmaticamente con l'operatore `throw`, è buona prassi propagare l'errore ricevuto nel caso fosse di diverso tipo da quello gestito nel blocco `try…catch`. Questa tecnica denominata «**rethrow**» è molto utile all'interno delle funzioni:

```
function evenNumber(num) {
  try {
    if (num % 2 !== 0) {
      throw new TypeError("The number must be even"). // Generiamo un errore se il numero non è pari.
    }
  } catch (error) {
    if (error instanceof TypeError)
      alert("Please, provide an even number!") // Qui gestiamo solamente l'errore del numero dispari.
    } else {
      throw error  // qualsiasi altro tipo di errore lo ri-propaghiamo all'esterno della funzione.
    }
  }
}
```

MDN Rethrow an exception

try...catch...finally

Dopo il blocco `catch` è possibile agganciare un'ulteriore istruzione, `finally`.

```
try {
  if (confirm("Make an error?")) {
    throw new Error("The user wants an error");
  }
} catch (e) {
  console.error("Shows only if user clicks OK");
} finally {
  console.log("Always shown");
}
```

L'istruzione `finally` è molto utile all'interno delle funzioni quando vogliamo comunque restituire un valore anche se il nostro codice ha generato un errore.

MDN Finally Block