Custom Exception Handling

**Custom exception handling using class**

Custom errors, extending Error

When we develop something, we often need our own error classes to reflect specific things that may go wrong in our tasks. For errors in network operations we may need HttpError, for database operations DbError, for searching operations NotFoundError and so on.

Our errors should support basic error properties like message, name and, preferably, stack. But they also may have other properties of their own, e.g. HttpError objects may have a statusCode property with a value like 404 or 403 or 500.

JavaScript allows to use throw with any argument, so technically our custom error classes don’t need to inherit from Error. But if we inherit, then it becomes possible to use obj instanceof Error to identify error objects. So it’s better to inherit from it.

As the application grows, our own errors naturally form a hierarchy. For instance, HttpTimeoutError may inherit from HttpError, and so on.

**Extending** **Error**

As an example, let’s consider a function readUser(json) that should read JSON with user data.

Here’s an example of how a valid json may look:

let json = `{ "name": "John", "age": 30 }`;

Internally, we’ll use JSON.parse. If it receives malformed json, then it throws SyntaxError. But even if json is syntactically correct, that doesn’t mean that it’s a valid user, right? It may miss the necessary data. For instance, it may not have name and age properties that are essential for our users.

Our function readUser(json) will not only read JSON, but check (“validate”) the data. If there are no required fields, or the format is wrong, then that’s an error. And that’s not a SyntaxError, because the data is syntactically correct, but another kind of error. We’ll call it ValidationError and create a class for it. An error of that kind should also carry the information about the offending field.

Example code

// The "pseudocode" for the built-in Error class defined by JavaScript itself

class Error {

constructor(message) {

this.message = message;

this.name = "Error"; // (different names for different built-in error classes)

this.stack = <call stack>; // non-standard, but most environments support it

}

}

this.message = message;

this.name = "Error"; // (different names for different built-in error classes)

this.stack = <call stack>; // non-standard, but most environments support it

}

}

//Now let’s inherit ValidationError from it and try it in action:

class ValidationError extends Error {

constructor(message) {

super(message); // (1)

this.name = "ValidationError"; // (2)

}

}

function test() {

throw new ValidationError("Whoops!");

}

try {

test();

} catch(err) {

alert(err.message); // Whoops!

alert(err.name); // ValidationError

alert(err.stack); // a list of nested calls with line numbers for each

}

More….

**Real life example**

Let’s try to use it in readUser(json):

class Error {

constructor(message) {

this.message = message;

this.name = "Error"; // (different names for different built-in error classes)

this.stack = <call stack>; // non-standard, but most environments support it

}

}

this.message = message;

this.name = "Error"; // (different names for different built-in error classes)

this.stack = <call stack>; // non-standard, but most environments support it

}

}

class ValidationError extends Error {

constructor(message) {

super(message);

this.name = "ValidationError";

}

}

// Usage

function readUser(json) {

let user = JSON.parse(json);

if (!user.age) {

throw new ValidationError("No field: age");

}

if (!user.name) {

throw new ValidationError("No field: name");

}

return user;

}

// Working example with try..catch

try {

let user = readUser('{ "age": 25 }');

} catch (err) {

if (err instanceof ValidationError) {

alert("Invalid data: " + err.message); // Invalid data: No field: name

} else if (err instanceof SyntaxError) { // (\*)

alert("JSON Syntax Error: " + err.message);

} else {

throw err; // unknown error, rethrow it (\*\*)

}

}

The try..catch block in the code above handles both our ValidationError and the built-in SyntaxError from JSON.parse.

Please take a look at how we use instanceof to check for the specific error type in the line (\*).

We could also look at err.name, like this:

// ...

// instead of (err instanceof SyntaxError)

} else if (err.name == "SyntaxError") { // (\*)

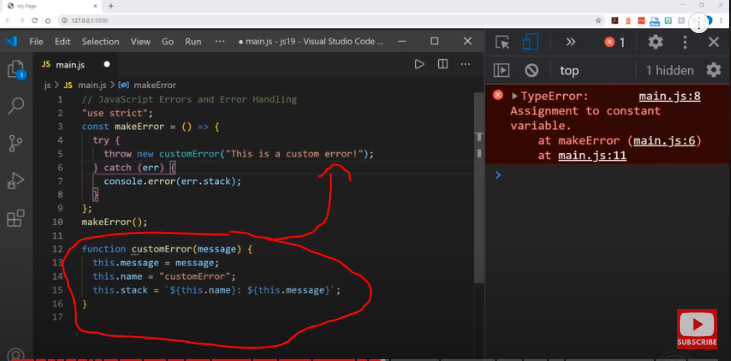
// ...

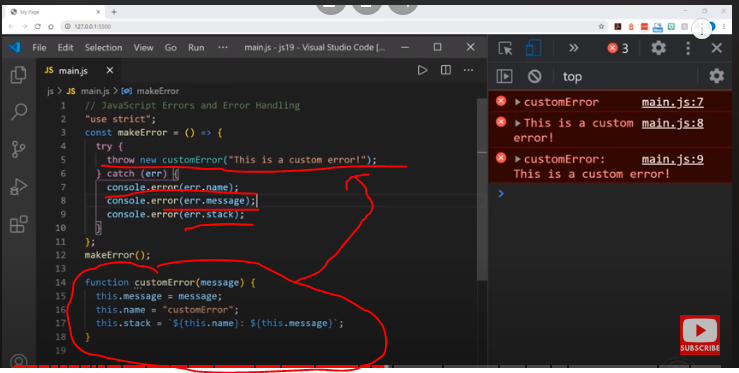
The instanceof version is much better, because in the future we are going to extend ValidationError, make subtypes of it, like PropertyRequiredError. And instanceof check will continue to work for new inheriting classes. So that’s future-proof.

Also it’s important that if catch meets an unknown error, then it rethrows it in the line (\*\*). The catch block only knows how to handle validation and syntax errors, other kinds (caused by a typo in the code or other unknown reasons) should fall through.

Link for more in-depth example: <https://javascript.info/custom-errors>

**Custom exception handling using functions**





another

<html>

<body>

<h2>JavaScript Error Handling</h2>

<p>How to use <b>catch</b> to display an error.</p>

<p id="demo"></p>

<script>

function error(message){

this.message = message;

this.name = "ReferenceError";

}

try {

//throw new error("Abbas"); // function call

// throw Error("error"); // call without new keyword

throw new Error("error2") // call with new keyword the result will same

altwert("Welcome guest!");

}

catch(err) {

alert(err.name);

alert(err.message);

}

</script>

</body>

</html>