Persistent

JavaScript: Error Handling

Persistent Interactive | Persistent University

Key learning points :

- Types of errors in JavaScript
- Try-catch block
- Error Object

Types of Errors

1. Standard JavaScript Errors
2. User defined Errors
3. System Errors
4. Assertion Errors

JavaScript Errors

- Some examples of JavaScript errors are Reference Error, Range Error, Eval Error, Syntax Error
- Thrown using standard JavaScript ‘throw’ mechanism.
- All JavaScript errors are handled as exceptions.
- These are handled with try-catch block.
- If any such errors are not handled with try-catch, then the Node process will exit immediately.

try- catch block

- An error inside the try {…} block does not kill the script: we have a chance to handle it in catch.

```
try{
    //An error can occur
}
}catch(e){

}
```

Example: Reference Error

- In the below code, z is not defined which will result in Reference Error.

```
// Throws with a ReferenceError because z is not defined.
try {
  const m = 1;
  const n = m + z;
} catch (err) {
  // Handle the error here.
}
```

Error Object

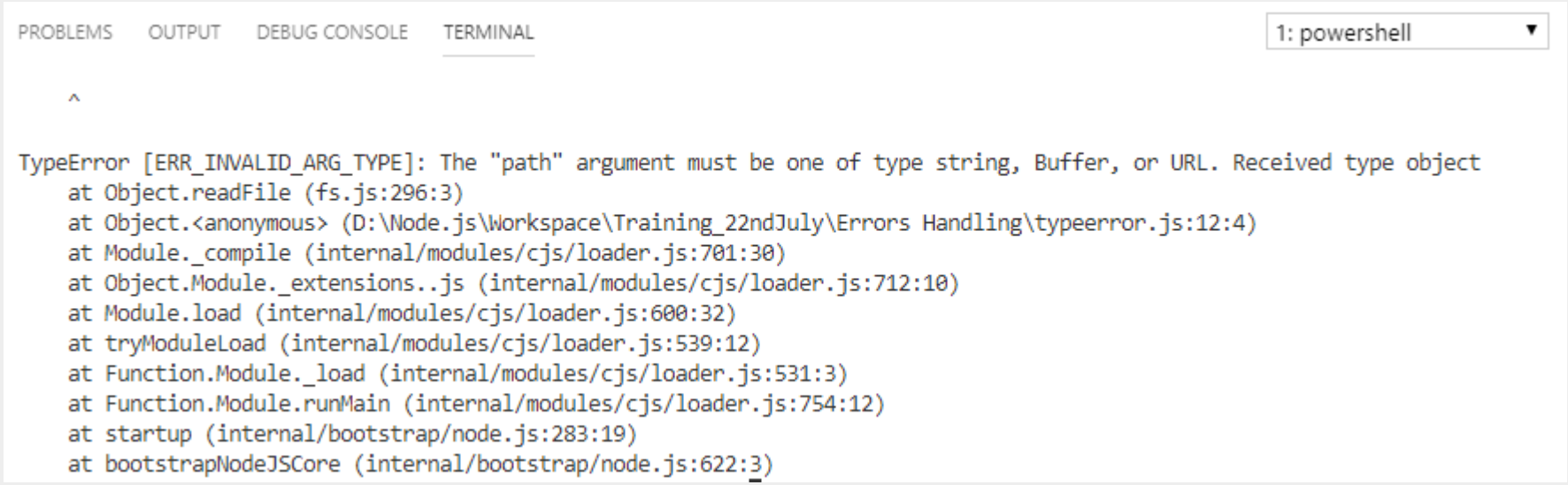
- When an error occurs, JavaScript generates an object containing the details about it. The object is then passed as an argument to catch.
- For all built-in errors, the error object inside catch block properties such as:

➢ **Name:** Error Name (Reference Error, Syntax Error)

➢ **Message:** Error description

➢ **Stack:** trace of which functions were called, in what order, from which line and file, and with what arguments.

Example



try..catch only works for runtime errors

- For try-catch to work, the code must be runnable i.e. it should be valid JavaScript.
- Such errors which can occur only during execution of valid JavaScript code, are called as **Runtime errors.**

```
try{
for(var i=0;i<5;i++){
            temp;    //Undeclared variable.
}
}catch(){
            console.error("Some Error occurred");
 }
```

Parse-time errors

- Try-catch wont work for code which has syntax errors.

```
try{
for(var i=0;i<5;i++){

//Unclosed for loop
}catch(){   //The catch won't be called  }
```

- The above code has an unclosed for loop. Such errors are termed as parse-time errors.

Syntax Errors

- This is a subclass of Error that indicates that a program is not valid JavaScript.
- These errors may only be generated and propagated as a result of code evaluation.
- Code evaluation may happen as a result of eval, Function, require.

Type Error

- A TypeError is thrown when an operand or argument passed to a function is incompatible with the type expected by that operator or function.

```
var fs = require('fs');

fs.readFile(null,function(){

})
```

- Here the first parameter is null instead of a string/buffer/url.
- Hence a TypeError will be thrown.

Error Class

- All errors generated by Node.js, including all System and JavaScript errors, will either be instances of Error class, or would inherit from it.
- Error objects capture a "stack trace" detailing the point in the code at which the Error was instantiated and may provide a text description of the error.
- The below code creates a new Error object and sets the **error.message** property to the provided text message.

```
var err = new Error(message)
var err = new Error(“User not found”);
```

Finally block

- Code inside a finally block runs in all cases irrespective of errors.
- It executes after try, if there were no errors,
- It executes after catch, even if there were errors.
- The finally clause works for any exit from try..catch. That includes an explicit return.

Persistent

Thank you!

Persistent Interactive | Persistent University