



Ejercicio 2: Otras herramientas para manejar errores

Materia: Computación tolerante a fallas

Universidad de Guadalajara

Profesor: Michel Emanuel López Franco

29/01/2024

José Maximiliano Díaz Méndez

Introducción

El manejo de los errores a la hora de diseñar sistemas es algo de gran importancia con lo cual conforme fueron evolucionando los lenguajes de programación incluyeron mejores herramientas para la creación y gestión de los mismos buscando hacer más fácil este proceso como se puede apreciar a continuación.

C#:

El manejo de errores en este lenguaje se hace mediante el uso de excepciones que representan el fallo en el sistema. Además de usarse los siguientes bloques de código para la gestión:

try: Dentro de este bloque va el código que es propenso a sufrir la falla y en caso de ocurrir el flujo de ejecución pasa al bloque **catch**.

catch: Aquí iría el código que se encargaría de gestionar el error, ya sea mostrando el error por consola o alguna otra acción.

finally: El código dentro de este bloque siempre se ejecutara independiente de si hubo un error o no por lo que se puede utilizar para liberar recursos u otras acciones. Este bloque es opcional para la gestión de errores.

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            Console.WriteLine("Enter a number: ");

            var num = int.parse(Console.ReadLine());

            Console.WriteLine($"Squire of {num} is {num * num}");
        }
        catch(Exception ex)
        {
            Console.Write("Error info: " + ex.Message);
        }
        finally
        {
            Console.Write("Re-try with a different number.");
        }
    }
}
```

JavaScript:

Similar a C#, JavaScript también maneja el uso de bloques **try**, **catch** y **finally** que facilita la gestión de errores. Salvo porque aquí en vez de usarse excepciones se usan errores que serían similares.


```
try {
  const number = 'a'
  if (typeof a !== 'number') {
    throw new Error('A number is expected')
  }
} catch (error) {
  console.error(error)
} finally {
  console.log('This code it\'s executed always')
}
```

Además del método anterior también es posible gestionar los errores de las **promesas**, un objeto especial en JavaScript que representa información que estará disponible en algún momento del futuro, pero no de forma inmediata, mediante el uso de **callbacks** con los métodos **catch**, **then** y **finally**. El problema de este método es que es fácil que suceda lo que vulgarmente se llama callback hell que hace bastante incomodo seguir el flujo del código.

```
const promise = new Promise((resolve, reject) => {
  if (typeof 'a' !== 'number') {
    reject(new Error('A number is expected'))
  } else {
    resolve('It\'s a number')
  }
})
promise
  .then((value) => { console.log(value) })
  .catch((error) => { console.error(error) })
  .finally(() => { console.log('This code it\'s always executed') })
```

Lua:

El manejo de errores en este lenguaje es mediante el uso de la función **error** para generar el error y la función **pcall** que recibe como primer parámetro la función que podría generar el error y retornando como primer valor un booleano indicando si ejecuto correctamente y como segundo el mensaje de error.



```
function foo ( )  
    error('Something happen')  
end  
local status, err = pcall(foo)  
if status then  
    print('Successful execution')  
else  
    io.stderr:write(err)  
end
```

Conclusión

La gran mayoría de los lenguajes de programación hoy en día ya cuentan con herramientas para hacer la gestión de errores de forma sencilla como se puede apreciar con los ejemplos anteriores usando herramientas que permiten ejecutar porciones de código que en caso de generar un error podemos atrapar para evitar su propagación inmediata y procesarlo.

Referencias

Exception handling in C#. (s. f.). Tutorials Teacher. Recuperado 25 de enero de 2024, de <https://www.tutorialsteacher.com/csharp/csharp-exception-handling>

Ierusalimschy, R. (s. f.). Programming in LUA : 8.4. Recuperado 26 de enero de 2024, de <https://www.lua.org/pil/8.4.html>

Mozilla foundation. (2023, 5 agosto). Try. . .Catch - JavaScript | MDN. MDN Web Docs. Recuperado 26 de enero de 2024, de <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/try...catch>

Mozilla Foundation. (2023, 8 agosto). Promise - JavaScript | MDN. MDN Web Docs. Recuperado 26 de enero de 2024, de https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Promise