Control de excepciones ES5

Este código define una función llamada BaseException que actúa como una especie de clase personalizada para manejar excepciones, basada en el objeto Error de JavaScript. Esencialmente, el código crea un nuevo tipo de excepción, llamado "MyError", que hereda de Error. Estos son los distintos componentes y su funcionamiento:

1. Función BaseException como "constructor" personalizado

```
function BaseException(message = "Default Message", fileName, lineNumber) {
    let instance = new Error(message, fileName, lineNumber);
    instance.name = "MyError";
    Object.setPrototypeOf(instance, Object.getPrototypeOf(this));
    if (Error.captureStackTrace) {
        Error.captureStackTrace(instance, BaseException);
    }
    return instance;
}
```

Desglose:
- Parámetros:
  - **message**: Mensaje opcional de error. Si no se proporciona, usa "Default Message".
  - **fileName y lineNumber**: Son parámetros opcionales que se pueden pasar para especificar el archivo y la línea en los que ocurrió el error, pero son opcionales en el constructor de Error.

- **let instance = new Error(...):**
  - Crea una nueva instancia del objeto Error estándar de JavaScript. Esta instancia es la base para nuestro error personalizado.

- **instance.name = "MyError":**
  - Cambia el nombre de la excepción a "MyError", en lugar de usar el nombre predeterminado ("Error").

- **Object.setPrototypeOf(instance, Object.getPrototypeOf(this)):**
  - Cambia el prototipo de instance al prototipo de this (el prototipo de la función BaseException). Esto asegura que instance herede correctamente de la excepción personalizada.

- Error.captureStackTrace:
  - Si está disponible (en entornos de JavaScript modernos como Node.js), captura el rastreo de la pila (stack trace) para que la traza de la excepción comience desde la llamada a BaseException, en lugar de desde la llamada a Error. Esto ayuda a depurar el error más fácilmente.

- return instance:

- En lugar de usar new explícitamente, devuelve la instancia creada manualmente, lo que permite que BaseException funcione como un "constructor".

2. Definición del prototipo

```
BaseException.prototype = Object.create(Error.prototype, {
    constructor: {
        value: BaseException,
        enumerable: false,
        writable: true,
        configurable: true
    }
});
```

Desglose:

- **BaseException.prototype = Object.create(Error.prototype, ...):**
  - Establece el prototipo de **BaseException** para que herede de **Error.prototype**. Esto significa que las instancias de **BaseException** tendrán las mismas propiedades y métodos que los errores estándar de JavaScript, como message y stack.

- **Propiedad constructor:**
  - Aquí se redefine la propiedad constructor de **BaseException** para que apunte a sí mismo. De manera predeterminada, después de usar **Object.create**, el constructor sería el de Error, pero este código asegura que el constructor sea **BaseException**.
  - Los atributos enumerable, writable y configurable se configuran explícitamente. Esto es común cuando se define una propiedad manualmente, para asegurarse de que el comportamiento sea controlable (por ejemplo, no ser enumerable para que no aparezca en bucles for-in).

Funcionamiento final:

Con este código, puedes crear y lanzar excepciones personalizadas de esta manera:

```
try {
    throw new BaseException("Something went wrong", "app.js", 10);
} catch (e) {
    console.log(e.name);    // "MyError"
    console.log(e.message); // "Something went wrong"
    console.log(e.stack);   // Stack trace que incluye la línea donde fue lanzado
}
```

¿Por qué hacerlo así?

Este patrón se usa para crear errores personalizados que:

1. Se comportan como errores estándar (tienen todas las propiedades y métodos de Error).
2. Tienen un nombre y comportamiento personalizado, lo que los hace útiles en situaciones donde se necesita categorizar o manejar errores específicos.
2. Capturan mejor el rastreo de pila (stack trace) para una depuración más precisa, manteniendo el contexto desde el origen del error.

InvalidAccessConstructorException

Este código define una nueva excepción personalizada llamada InvalidAccessConstructorException, que extiende la funcionalidad de la excepción base BaseException, definida previamente.

Vamos a desglosarlo:

1. Función InvalidAccessConstructorException

```
function InvalidAccessConstructorException() {
    let instance = BaseException.call(this, "Constructor can’t be called as a function.");
    instance.name = "InvalidAccessConstructorException";
    return instance;
}
```

Desglose:

- BaseException.call(this, "Constructor can’t be called as a function.");:
  - Esta línea llama al constructor BaseException, pero usando call. Al hacer esto, se pasa el contexto actual (this) a BaseException, lo que significa que la nueva instancia de InvalidAccessConstructorException heredará las propiedades y métodos definidos en BaseException.
  - Además, pasa el mensaje "Constructor can’t be called as a function." como argumento, lo que se convertirá en el mensaje de error.
  - Básicamente, esta línea reutiliza la lógica de BaseException para configurar las propiedades básicas de la excepción (como el mensaje y el nombre).

- **instance.name = "InvalidAccessConstructorException";:**
  - Cambia el nombre de la excepción a "InvalidAccessConstructorException". Esto ayuda a diferenciarla de otros tipos de excepciones cuando se manejen los errores.

- return instance;:
  - Devuelve la instancia creada por BaseException. Este enfoque permite que el constructor devuelva una instancia de la excepción correctamente configurada.

2. Herencia del prototipo

```
InvalidAccessConstructorException.prototype = Object.create(BaseException.prototype);
InvalidAccessConstructorException.prototype.constructor =
InvalidAccessConstructorException;
```

Desglose:

- **InvalidAccessConstructorException.prototype = Object.create(BaseException.prototype);:**
  - Se configura el prototipo de InvalidAccessConstructorException para que herede del prototipo de BaseException. Esto asegura que todas las instancias de InvalidAccessConstructorException tengan acceso a las propiedades y métodos definidos en BaseException (como el mensaje, nombre y stack trace).

- **InvalidAccessConstructorException.prototype.constructor = InvalidAccessConstructorException;:**
  - Aquí se restablece la propiedad constructor para que apunte a **InvalidAccessConstructorException**. Esto es importante porque al sobrescribir el prototipo con Object.create, la propiedad constructor apuntaría a **BaseException**, lo cual no es correcto. Este paso garantiza que el constructor refleje correctamente la nueva clase de error.

Funcionamiento del código

El código crea una nueva clase de excepción llamada InvalidAccessConstructorException que extiende la funcionalidad de BaseException. **Esta excepción se lanza cuando ocurre una situación específica, en este caso, cuando un constructor es llamado incorrectamente como una función**. El mensaje de error predeterminado es "Constructor can't be called as a function.".

Ejemplo de uso

Podrías usar esta excepción en un caso donde quieras prevenir que los constructores sean llamados sin la palabra clave new:

```
function SomeConstructor() {
    if (!(this instanceof SomeConstructor)) {
        throw new InvalidAccessConstructorException();
    }
    // Resto del constructor...
}

try {
    SomeConstructor(); // Error porque no se usa 'new'
} catch (e) {
    console.log(e.name);    // "InvalidAccessConstructorException"
    console.log(e.message); // "Constructor can't be called as a function."
}
```

Resumen

- InvalidAccessConstructorException es una excepción personalizada que hereda de BaseException.
- Se utiliza para manejar específicamente el error cuando un constructor es llamado como una función en lugar de ser instanciado con new.
- Hereda todas las propiedades y métodos de BaseException, incluyendo la capacidad de tener un mensaje de error personalizado y capturar el stack trace.