JAVARUSH

Путь к карьере Frontend Fullstack разработчика

Модуль 1. WEB CORE

Уровень 18. Функции, массивы и объекты.
Часть 3.

Внештатные ситуации

При работе любой программы могут возникать внештатные ситуации.

Программист (автор программы) должен каждую внештатную ситуацию:

- предугадать
- решить, как именно программа должна работать в этой ситуации
- запрограммировать решение, максимально близкое к желаемому.

JAVARUSH

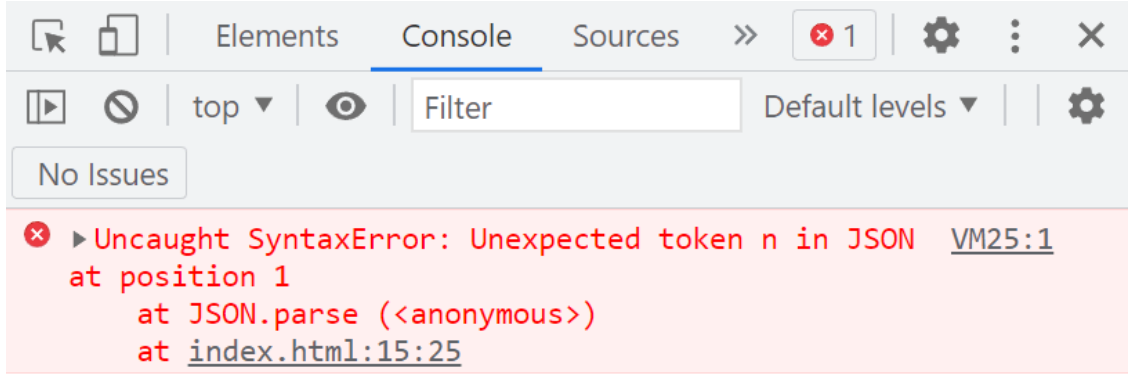
Что такое исключение

В JavaScript исключение — это событие, которое прерывает нормальное выполнение программы и сообщает, что произошла ошибка. Исключения могут возникать по разным причинам:

- Ошибки в синтаксисе
- Неправильное использование функций
- Попытки обращения к несуществующим объектам или свойствам
- Другие ошибки, связанные с логикой программы

Исключением может быть строка, число, логическое значение или объект.

Когда возникает ошибка, **выполнение кода прекращается**, и эта ошибка выводится в консоль:



JAVARUSH

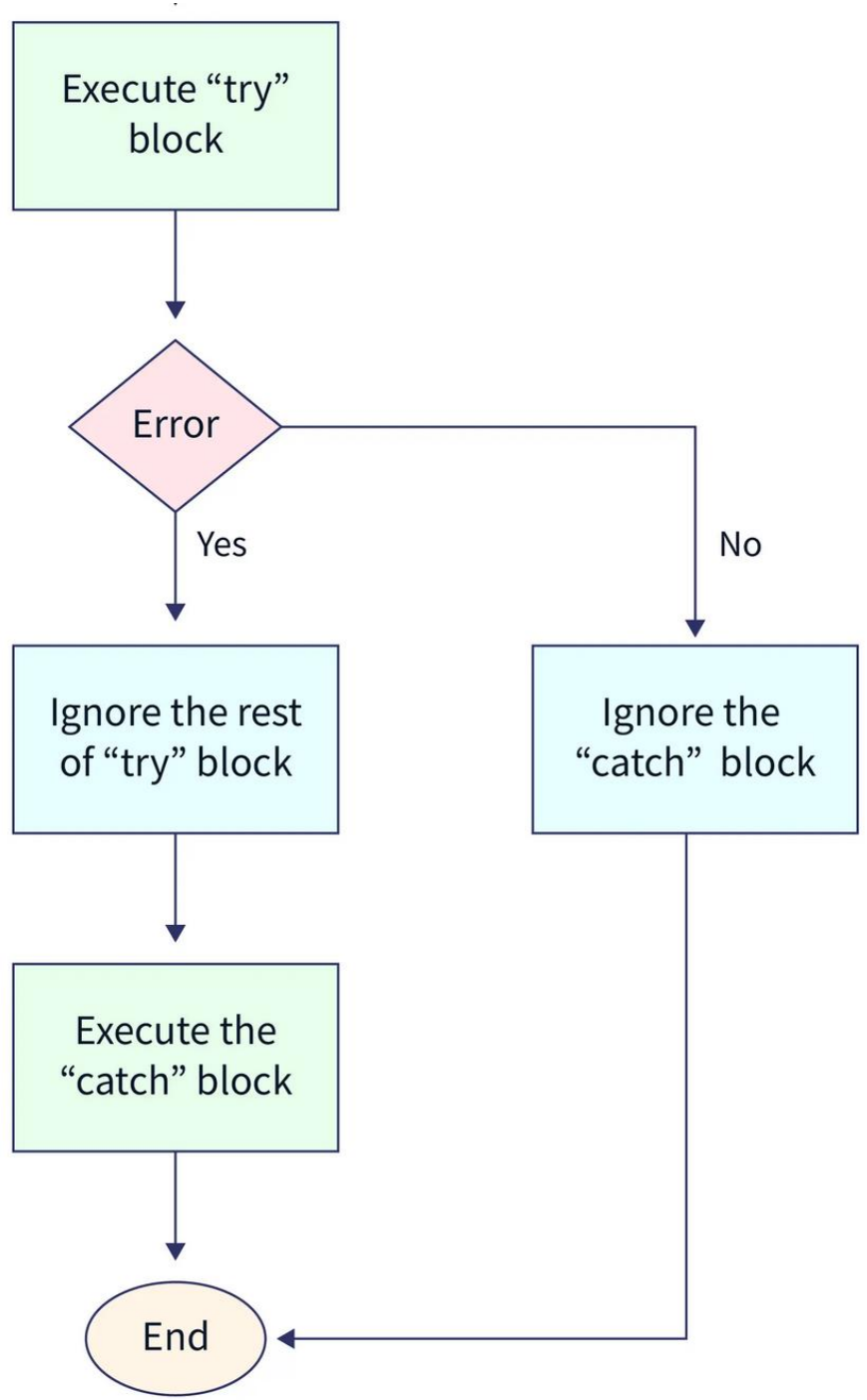
Блок try...catch

Конструкция try...catch используется для обработки ошибок, возникающих в блоке кода try. Если в блоке try возникает ошибка, управление передается в блок catch, где ошибка может быть обработана.

```
try {
    // Код, который может вызвать исключение
} catch(error) {
    // Код для обработки исключения
}
```

```
try {
    // Код, который может вызвать ошибку
    let result = 10 / 0; // Вызовет ошибку деления на ноль
} catch (error) {
    console.error('Произошла ошибка:', error.message);
}
```

Блок try: Здесь мы помещаем код, который может вызвать ошибку.
Блок catch: Если в блоке try возникает ошибка, выполнение переходит сюда. Параметр error содержит информацию об ошибке.



JAVARUSH

finally: Гарантированное завершение

finally – это ключевое слово в JavaScript, которое используется для выполнения кода, который должен быть выполнен независимо от того, возникло исключение в блоке try или нет.

У кода есть два пути выполнения:

1. Если вы ответите на вопрос «Сгенерировать ошибку?» утвердительно, то try -> catch -> finally.
2. Если ответите отрицательно, то try -> finally.

```
try {
   // Код, который может вызвать ошибку
} catch (error) {
   // Обработка ошибки
} finally {
   // Этот код выполнится всегда
}
```

```
let result = 0;
try {
   result = sum(10, 20);
// Предположим, что функция sum не определена
   console.log('Это сообщение мы не увидим!');
} catch (error) {
   console.log(error.message);
} finally {
   console.log('Результат:', result);
}
```

JAVARUSH

try...finally

finally можно использовать и без блока catch. Секцию finally часто используют, когда мы начали что-то делать и хотим завершить это вне зависимости от того, будет ошибка или нет.

```
try {
   // Отправить данные на сервер, здесь нам неважна обработка ошибки
   sendData()
} finally {
   // Закрыть соединение при любом результате
   closeConnection()
}
```

```
let mysql = require('mysql');
let connection = mysql.createConnection({/* ... */});
connection.connect();

try {
   // Выполняем SQL-запросы
} finally {
   connection.end();
   console.log('Соединение с базой данных закрыто');
}
```

JAVARUSH

Оператор throw

Оператор throw используется для явного выбрасывания исключений. Он дает нам возможность создать собственные ошибки и контролировать их обработку.

Оператор throw используют в тех случаях, когда нужно указать на ошибку, например внутри функций или условий. При использовании throw выполнение кода немедленно прерывается.

```
function divide(a, b) {
    if (b === 0) {
        throw new Error('Деление на ноль запрещено!');
    }
    return a / b;
}

try {
    let result = divide(10, 0);
    console.log(result);
} catch (error) {
    console.error('Произошла ошибка:', error.message);
}
```

JAVARUSH

Объект Error

Когда возникает ошибка, JavaScript создает объект Error, который содержит информацию о ней:

- **message:** Текстовое описание ошибки.
- **name:** Тип ошибки (например, "ReferenceError").
- **stack:** Стековый след, показывающий, где произошла ошибка.

```
try {
    let x = y + 1; // y не определена
} catch (error) {
    console.error(error.name); // ReferenceError
    console.error(error.message); // y is not defined
    console.error(error.stack); // Stack
}
```

```
// Создание пользовательских ошибок.
// Класс CustomError наследуется от базового класса Error,
class CustomError extends Error {
    constructor(message) {
        super(message); // Вызываем конструктор родительского класса
        this.name = 'CustomError';
    }
}

function validateAge(age) {
    if (age < 0) {
        // Создаем новый объект ошибки CustomError с указанным сообщением.
        // Оператор throw прерывает выполнение функции и передает управление
        throw new CustomError('Возраст не может быть отрицательным');
    }
}
```

JAVARUSH

Встроенные ошибки

SyntaxError: Ошибка в синтаксисе кода (например, забыли поставить точку с запятой).

ReferenceError: Обращение к несуществующей переменной или свойству.

TypeError: Попытка выполнить операцию с недопустимым типом данных.

RangeError: Значение выходит за допустимые пределы (например, отрицательный индекс массива).

EvalError: Ошибка, связанная с функцией eval().

URIError: Ошибка, связанная с кодированием URI.

CustomError: Создаваемые нами ошибки для более специфичных ситуаций.

JAVARUSH

Вложенные блоки try...catch

Иногда для более гибкой обработки ошибок требуется использовать несколько уровней блоков try...catch. Это позволяет ловить ошибки на разных этапах выполнения кода и реагировать на них по-разному.

Как работают вложенные блоки try...catch:

- **Внутренний блок:** Обрабатывает ошибки, возникающие внутри него.
- **Внешний блок:** Обрабатывает ошибки, возникающие как внутри, так и снаружи внутреннего блока.
- **Блок finally:** Выполняется всегда, независимо от того, произошла ошибка или нет.

```
try {
    try {
        // Код, который может вызвать ошибку
        let result = divide(10, 0);
        console.log(result);
    } catch (innerError) {
        console.error('Внутренняя ошибка:', innerError.message);
        // Перебрасываем ошибку наружу для дальнейшей обработки
        throw innerError;
    }
} catch (outerError) {
    console.error('Внешняя ошибка:', outerError.message);
} finally {
    console.log('Выполнение вложенных блоков try...catch
завершено.');
}
```

JAVARUSH

Домашнее задание

Уровень 18. Функции, массивы и объекты