JavaScript Hibakezelés Magyarázata

A hibakezelés fontos része minden programozási nyelvnek, beleértve a JavaScriptet is. Lehetővé teszi, hogy a programozók kezeljék a váratlan helyzeteket, például a felhasználói hibákat, a hálózati hibákat vagy a programozási hibákat. A JavaScript hibakezelési mechanizmusa a `try...catch` blokkot használja.

Hibakezelési Mechanizmus

try...catch Blokk

A `try...catch` blokk két részből áll:

- **try blokk**: Itt helyezzük el azt a kódot, amely esetleg hibát okozhat.
- **catch blokk**: Ez a blokk fut le, ha a `try` blokkban hiba történik. Itt kezeljük a hibát.

Szintaxis:

```
try {
    // Code that may throw an error
} catch (error) {
    // Code to handle the error
}
```

Példa: Egyszerű Hibakezelés

```
try {
    let result = riskyOperation();
    console.log(result);
} catch (error) {
    console.error("An error occurred:", error.message);
}
```

Kiegészítő `finally` Blokk

A `finally` blokk egy opcionális harmadik rész a hibakezelési struktúrában, amely akkor is végrehajtódik, ha hiba történt, és akkor is, ha nem. A finally akkor is lefut, ha a try blockban return történt.

Szintaxis:

```
try {
    // Code that may throw an error
} catch (error) {
    // Code to handle the error
} finally {
    // Code that will always run, regardless of an error
}
```

Példa: try...catch...finally

```
try {
    let result = riskyOperation();
    console.log(result);
} catch (error) {
    console.error("An error occurred:", error.message);
} finally {
    console.log("This will always run, regardless of an error.");
}
```

Hibák Dobása (Throwing Errors)

A `throw` kulcsszó használatával saját hibákat is dobhatunk. Ez különösen hasznos, ha egy függvényben egyedi hibákat szeretnénk kezelni.

Szintaxis:

```
throw new Error("Something went wrong");
```

Példa: Hibák Dobása

```
function divide(a, b) {
    if (b === 0) {
        throw new Error("Division by zero is not allowed");
    }
    return a / b;
}

try {
    let result = divide(10, 0);
    console.log(result);
} catch (error) {
    console.error("An error occurred:", error.message);
}
```

Egyedi Hibatípusok

A JavaScript rendelkezik néhány beépített hibatípussal, mint például `Error`, `TypeError`, `RangeError`, `SyntaxError`, stb. Ezeket a hibatípusokat használhatjuk egyedi hibák dobására.

Példa: Egyedi Hibatípusok Használata

```
function checkArray(arr) {
    if (!Array.isArray(arr)) {
        throw new TypeError("Expected an array");
    }
    if (arr.length === 0) {
        throw new RangeError("Array cannot be empty");
    }
    return arr.length;
}

try {
    let result = checkArray(123);
    console.log(result);
} catch (error) {
    console.error(`${error.name}: ${error.message}`);
}
```

Böngésző mint Host Environment

Böngésző mint Host Environment

A JavaScript nem csak a böngészőben futtatható, de a böngésző az egyik leggyakoribb környezet, ahol használják. A böngésző, mint "host environment" (gazda környezet), a JavaScript motor számára biztosít egy környezetet, amelyben a kód futtatható. Ez a környezet számos beépített objektumot és függvényt tartalmaz, amelyek segítségével a JavaScript hozzáférhet a weboldal elemeihez és a böngésző funkcióihoz.

BOM (Browser Object Model)

Mi az a BOM?

A BOM (Browser Object Model) egy gyűjteménye azoknak az objektumoknak és függvényeknek, amelyeket a böngésző biztosít a JavaScript számára a böngészőablak és a weblap közötti interakció kezeléséhez. A BOM nem része a szabványosított JavaScript-nek, de a legtöbb böngésző támogatja.

A BOM objektumok lehetővé teszik például:

- Az ablak tulajdonságainak és méretének lekérdezését és módosítását.
- Az aktuális URL lekérdezését és módosítását.
- A böngésző történetének kezelését.
- Információk lekérdezését a böngészőről és a felhasználó képernyőjéről.

Példa a BOM használatára

```
// Az ablak szélességének és magasságának lekérése
let width = window.innerWidth;
let height = window.innerHeight;

console.log(`Szélesség: ${width}, Magasság: ${height}`);
```

Window Object

Mi az a Window Object?

A `window` objektum a BOM központi objektuma, amely a böngészőablakot reprezentálja. Minden, a BOM-ban lévő objektum, függvény és változó a `window` objektum része. Ezért a `window` objektum nélkülözhetetlen része a böngésző környezetének, és a JavaScript számára egy globális kontextust biztosít.

A `window` Objektum Tulajdonságai és Metódusai

- **`window.document`**: A jelenlegi HTML dokumentumot reprezentálja, és hozzáférést biztosít a DOM-hoz.
- **`window.location`**: Az aktuális URL-t kezeli, és lehetővé teszi annak módosítását.
- **`window.navigator`**: Információkat ad a böngészőről és az operációs rendszerről.
- **`window.history`**: A böngésző történetét kezeli, lehetővé téve az előre és hátra navigációt.
- **`window.screen`**: Információkat ad a felhasználó képernyőjéről.
- **`window.innerWidth` és `window.innerHeight`**: Az ablak belső szélessége és magassága.
- **`alert()`, `confirm()`, `prompt()`**: Felugró ablakokat jelenít meg.

Példa a `window` Objektum Használatára

```
// URL módosítása
window.location.href = "https://www.example.com";

// Információk lekérése a böngészőről
console.log(window.navigator.userAgent);

// Felugró ablak megjelenítése
window.alert("Hello, World!");

// Ablak méreteinek lekérése
let width = window.innerWidth;
let height = window.innerHeight;

console.log(`Szélesség: ${width}, Magasság: ${height}`);
```

A DOM (Document Object Model)

Mi az a DOM?

A DOM (Document Object Model) egy programozási interfész a HTML és XML dokumentumokhoz. Lehetővé teszi, hogy a dokumentumok szerkezete fára épülő módon legyen reprezentálva, ahol minden csomópont egy dokumentum része, mint például elemek, attribútumok, szövegek, stb.

A DOM segítségével a JavaScript:

- Hozzáférhet a HTML dokumentum struktúrájához és tartalmához.
- Módosíthatja a HTML dokumentum szerkezetét, stílusait és tartalmát.
- Eseménykezelőket adhat hozzá az elemekhez, például kattintások, billentyűleütések kezelése.

DOM Fa Struktúra

A DOM fa egy hierarchikus szerkezet, amely csomópontokat (node) tartalmaz. Minden csomópont egy elemet, attribútumot, szöveget vagy más dokumentumot reprezentál. A csomópontok típusai a következők:

- **Elem csomópontok**: HTML elemeket reprezentálnak (pl. `<div>`, `<p>`, `<a>`).
- **Attribútum csomópontok**: Az elemek attribútumait reprezentálják (pl. `id`, `class`).
- **Szöveg csomópontok**: Az elemek belső szövegét reprezentálják.
- **Komment csomópontok**: HTML kommenteket reprezentálnak (pl. `<!-- Comment -->`).
- **Dokumentum csomópontok**: A teljes dokumentumot reprezentálják.

DOM Fa Példa

Vegyünk egy egyszerű HTML dokumentumot, és nézzük meg, hogyan néz ki a DOM fa struktúrája.

HTML Dokumentum

```
<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
</head>
<body>
    <h1 id="main-title">Hello, World!</h1>
    <p>This is a paragraph.</p>
</body>
</html>
```

DOM Fa Struktúra

```
Document
 └── html
      ├── head
      │    └── title
      │         └── "Example"
      └── body
           ├── h1 (id="main-title")
           │    └── "Hello, World!"
           └── p
                └── "This is a paragraph."
```

DOM Csomópontok

Dokumentum Csomópont (Document Node)

A legfelső szintű csomópont, amely az egész HTML vagy XML dokumentumot reprezentálja.

Elem Csomópont (Element Node)

Minden HTML elem egy elem csomópont. Például a `<body>`, `<h1>`, és `<p>` elemek mind elem csomópontok.

Attribútum Csomópont (Attribute Node)

Az elem csomópontok attribútumai. Például az `id="main-title"` attribútum egy attribútum csomópont.

Szöveg Csomópont (Text Node)

Az elem csomópontok szöveges tartalmát reprezentálják. Például a `<h1>` elemben a "Hello, World!" egy szöveg csomópont.

DOM Fa Navigáció

A DOM fa csomópontjain különböző módokon navigálhatunk.

Navigációs Tulajdonságok

- **`parentNode`**: A szülő csomópont.
- **`parentElement`**: A szülő elem.
- **`childNodes`**: Az összes gyermek csomópont collection-je.
- **`children`**: Az összes gyermek elem collection-je.
- **`firstChild`**: Az első gyermek csomópont.
- **`firstElementChild`**: Az első gyermek elem.
- **`lastChild`**: Az utolsó gyermek csomópont.
- **`lastElementChild`**: Az utolsó gyermek elem.
- **`previousSibling`**: Az előző testvér csomópont.
- **`previousElementSibling`**: Az előző testvér elem.
- **`nextSibling`**: A következő testvér csomópont.
- **`nextElementSibling`**: A következő testvér elem.

Példa: DOM Navigáció

```
// Hozzáférés a <body> szülő eleméhez (ami a <html>)
let bodyParent = document.body.parentNode;
console.log(bodyParent.nodeName); // Output: "HTML"

// Hozzáférés a <body> első gyermekéhez
let firstChild = document.body.firstChild;
console.log(firstChild.nodeName); // Output: (valószínűleg text node, ha van whitespace)
```