07.08.

Aszinkron JavaScript

Mi az az Aszinkron JavaScript?

Az aszinkron programozás lehetővé teszi, hogy a JavaScript kód blokkolás nélkül végezzen el műveleteket. Ez különösen hasznos, amikor hosszabb időt igénylő műveleteket, például hálózati kéréseket vagy időigényes számításokat kell végezni, mivel lehetővé teszi, hogy a program más műveleteket is végrehajtson ahelyett, hogy várna ezeknek a műveleteknek a befejezésére.

Callback Hell és Pyramid of Doom

Callback Hell

A "callback hell" egy olyan helyzet, amikor mélyen egymásba ágyazott callback-eket használunk, amelyek miatt a kód olvashatatlanná és karbantarthatatlanná válik.

Példa Callback Hell-re:

```
fs.readFile("file1.txt", "utf8", function (err, data1) {
    if (err) {
        throw err;
    }
    fs.readFile("file2.txt", "utf8", function (err, data2) {
        if (err) {
            throw err;
        }
        fs.readFile("file3.txt", "utf8", function (err, data3) {
            if (err) {
                throw err;
            }
            console.log(data1, data2, data3);
        });
    });
});
```

Pyramid of Doom

A "pyramid of doom" kifejezés arra a helyzetre utal, amikor a kódunk egymásba ágyazott callback-ek miatt piramis alakot vesz fel, ami megnehezíti az olvasást és a hibakeresést.

Példa Pyramid of Doom-ra:

```
doSomething(function (result) {
    doSomethingElse(result, function (newResult) {
        doAnotherThing(newResult, function (finalResult) {
            console.log(finalResult);
        });
    });
});
```

Promise-ok

Promise Szintaxis

A Promise egy objektum, amely egy aszinkron művelet végkimenetelét reprezentálja. Egy Promise három állapotban lehet: pending, fulfilled, rejected.

Promise Létrehozása

```
let myPromise = new Promise(function(resolve, reject) {
    // Aszinkron művelet
    if (/* sikeres */) {
        resolve("Siker!");
    } else {
        reject("Hiba történt");
    }
});
```

Promise Állapotok

- **Pending**: A művelet folyamatban van.
- **Fulfilled**: A művelet sikeresen befejeződött, és egy értéket adott vissza.
- **Rejected**: A művelet sikertelen volt, és egy hibát adott vissza.

resolve és reject

- **resolve**: Meghívásakor a Promise állapota `fulfilled` lesz, és a megadott értéket adja vissza.
- **reject**: Meghívásakor a Promise állapota `rejected` lesz, és a megadott hibát adja vissza.

Példa resolve és reject használatára:

```
let myPromise = new Promise(function (resolve, reject) {
    setTimeout(function () {
        let success = true; // Ez egy példa, valós helyzetben itt történne valami aszinkron művelet
        if (success) {
            resolve("Siker!");
        } else {
            reject("Hiba történt");
        }
    }, 2000);
});
```

then, catch, finally

- **then**: A Promise sikeres teljesülésekor (fulfilled) fut le, és az eredményt adja vissza.
- **catch**: A Promise elutasítása (rejected) esetén fut le, és a hibát adja vissza.
- **finally**: Függetlenül attól, hogy a Promise sikeresen teljesült vagy elutasításra került, mindig lefut.

Példa then, catch, finally használatára:

```
myPromise
    .then(function (value) {
        console.log("Siker:", value);
    })
    .catch(function (error) {
        console.log("Hiba:", error);
    })
    .finally(function () {
        console.log("Ez mindig lefut, függetlenül a Promise állapotától.");
    });
```

Promise-ok Láncolása

A Promise-ok láncolása lehetővé teszi több aszinkron művelet sorozatos végrehajtását, mindegyik a korábbi művelet eredményére építve. A `then` metódus mindig új Promise-t ad vissza, amely lehetővé teszi a láncolást.

Példa Promise-ok Láncolására:

```
let firstPromise = new Promise(function (resolve, reject) {
    setTimeout(function () {
        resolve("Első művelet kész!");
    }, 1000);
});

firstPromise
    .then(function (value) {
        console.log(value);
        return new Promise(function (resolve, reject) {
            setTimeout(function () {
                resolve("Második művelet kész!");
            }, 1000);
        });
    })
    .then(function (value) {
        console.log(value);
        return new Promise(function (resolve, reject) {
            setTimeout(function () {
                resolve("Harmadik művelet kész!");
            }, 1000);
        });
    })
    .then(function (value) {
        console.log(value);
    })
    .catch(function (error) {
        console.error(error);
    });
```