Statikus Promise metódusok

Promise.all

A `Promise.all` metódus egyetlen Promise-t ad vissza, amely akkor teljesül, ha az összes megadott Promise teljesül, vagy elutasításra kerül, ha bármelyik Promise elutasításra kerül. Az eredmény egy tömb lesz, amely az összes Promise értékeit tartalmazza.

Szintaxis:

```
Promise.all([promise1, promise2, ...]);
```

Példa:

```
let promise1 = Promise.resolve(3);
let promise2 = 42;
let promise3 = new Promise((resolve, reject) => {
    setTimeout(resolve, 100, "foo");
});

Promise.all([promise1, promise2, promise3])
    .then((values) => {
        console.log(values); // [3, 42, "foo"]
    })
    .catch((error) => {
        console.error(error);
    });
```

Promise.allSettled

A `Promise.allSettled` metódus egyetlen Promise-t ad vissza, amely akkor teljesül, amikor az összes megadott Promise teljesül vagy elutasításra kerül. Az eredmény egy tömb lesz, amely minden Promise állapotát és értékét vagy hibáját tartalmazza.

Szintaxis:

```
Promise.allSettled([promise1, promise2, ...]);
```

Példa:

```
let promise1 = Promise.resolve(3);
let promise2 = new Promise((resolve, reject) => {
    setTimeout(reject, 100, "foo");
});

Promise.allSettled([promise1, promise2]).then((results) => {
    results.forEach((result) =>
        console.log(result.status, result.value || result.reason)
    );
    // "fulfilled" 3
    // "rejected" "foo"
});
```

Promise.race

A `Promise.race` metódus egyetlen Promise-t ad vissza, amely akkor teljesül vagy kerül elutasításra, amikor az első megadott Promise teljesül vagy elutasításra kerül. Az eredmény az első befejezett Promise értéke vagy hibája lesz.

Szintaxis:

```
Promise.race([promise1, promise2, ...]);
```

Példa:

```
let promise1 = new Promise((resolve, reject) => {
    setTimeout(resolve, 500, "one");
});
let promise2 = new Promise((resolve, reject) => {
    setTimeout(resolve, 100, "two");
});

Promise.race([promise1, promise2])
    .then((value) => {
        console.log(value); // "two"
    })
    .catch((error) => {
        console.error(error);
    });
```

Promise.any

A `Promise.any` metódus egyetlen Promise-t ad vissza, amely akkor teljesül, ha az első Promise teljesül. Ha az összes megadott Promise elutasításra kerül, akkor az eredmény egy aggregált hiba lesz.

Szintaxis:

```
Promise.any([promise1, promise2, ...]);
```

Példa:

```
let promise1 = new Promise((resolve, reject) => {
    setTimeout(reject, 100, "foo");
});
let promise2 = new Promise((resolve, reject) => {
    setTimeout(resolve, 500, "bar");
});
let promise3 = new Promise((resolve, reject) => {
    setTimeout(resolve, 1000, "baz");
});

Promise.any([promise1, promise2, promise3])
    .then((value) => {
        console.log(value); // "bar"
    })
    .catch((error) => {
        console.error(error);
    });
```

Promise.resolve

A `Promise.resolve` metódus egy új, teljesült Promise-t ad vissza a megadott értékkel. Ha az érték egy Promise, akkor az új Promise az eredeti Promise állapotát fogja tükrözni.

Szintaxis:

```
Promise.resolve(value);
```

Példa:

```
let resolvedPromise = Promise.resolve(42);

resolvedPromise.then((value) => {
    console.log(value); // 42
});
```

Promise.reject

A `Promise.reject` metódus egy új, elutasított Promise-t ad vissza a megadott hibával vagy okkal.

Szintaxis:

```
Promise.reject(reason);
```

Példa:

```
let rejectedPromise = Promise.reject(new Error("Hiba történt"));

rejectedPromise.catch((error) => {
    console.error(error); // Error: Hiba történt
});
```

Hálózati Kérés (Network Request)

Mi az a Hálózati Kérés?

A hálózati kérés egy olyan folyamat, amely során egy számítógép (kliens) adatokat kér egy másik számítógéptől (szerver) a hálózaton keresztül. A webfejlesztésben a hálózati kéréseket gyakran használják adatok lekérésére vagy küldésére egy szerverre az interneten keresztül.

Példa Hálózati Kérésre:

Amikor egy böngésző megnyit egy weboldalt, egy hálózati kérést küld a weboldal szerverére, amely visszaküldi a weboldal HTML kódját. Ez a HTML kód letöltésre kerül, és a böngésző megjeleníti azt.

API (Application Programming Interface)

Mi az az API?

Az API (Application Programming Interface) egy olyan interfész, amely lehetővé teszi, hogy két különböző szoftveralkalmazás kommunikáljon egymással. Az API meghatározza a szabályokat és protokollokat, amelyeken keresztül az alkalmazások adatokat cserélhetnek.

Példa API-ra:

Egy időjárás API lehetővé teszi, hogy egy alkalmazás lekérje az aktuális időjárási adatokat egy adott helyszínről. Az alkalmazás HTTP kéréseket küld az API-nak, és az API visszaküldi az időjárási adatokat JSON formátumban.

HTTP Metódusok

Mi az a HTTP?

A HTTP (HyperText Transfer Protocol) az alapvető protokoll, amelyet a weben használnak az adatok átvitelére. A HTTP metódusok meghatározzák, hogy milyen műveleteket lehet végrehajtani az erőforrásokon.

Gyakori HTTP Metódusok

- **GET**: Adatok lekérése egy szerverről. A GET kéréseknek nincs mellékhatása, azaz nem módosítják a szerveren lévő adatokat.
- **POST**: Adatok küldése a szerverre. A POST kéréseket gyakran használják adatok beküldésére, például űrlapok küldésére.
- **PUT**: Egy meglévő erőforrás módosítása vagy egy új erőforrás létrehozása a szerveren.
- **DELETE**: Egy erőforrás törlése a szerverről.
- **PATCH**: Egy meglévő erőforrás részleges módosítása a szerveren.

Példa GET Kérésre:

```
GET /users HTTP/1.1
Host: api.example.com
```

JSON (JavaScript Object Notation)

Mi az a JSON?

A JSON (JavaScript Object Notation) egy könnyű adatcsere formátum, amelyet könnyű ember által olvasni és írni, valamint gépek számára elemezni és generálni. A JSON-t gyakran használják API-kban az adatok cseréjére.

JSON Példa:

```
{
    "name": "John Doe",
    "age": 30,
    "city": "New York"
}
```

fetch Használata

Mi az a fetch?

A `fetch` egy modern JavaScript API, amely lehetővé teszi hálózati kérések egyszerű végrehajtását. A `fetch` segítségével aszinkron HTTP kéréseket küldhetünk és fogadhatunk adatokat a szerverről.

fetch Szintaxis

Alapvető GET Kérés:

```
fetch("https://api.example.com/data")
    .then((response) => {
        if (!response.ok) {
            throw new Error("Hálózati hiba történt");
        }
        return response.json();
    })
    .then((data) => {
        console.log(data);
    })
    .catch((error) => {
        console.error("Hiba:", error);
    });
```

POST Kérés:

```
fetch("https://api.example.com/data", {
    method: "POST",
    headers: {
        "Content-Type": "application/json",
    },
    body: JSON.stringify({ name: "John Doe", age: 30 }),
})
    .then((response) => {
        if (!response.ok) {
            throw new Error("Hálózati hiba történt");
        }
        return response.json();
    })
    .then((data) => {
        console.log(data);
    })
    .catch((error) => {
        console.error("Hiba:", error);
    });
```

fetch Paraméterek

- **URL**: A kért erőforrás URL-je.
- **options**: Egy opcionális objektum, amely tartalmazza a kéréssel kapcsolatos beállításokat (pl. metódus, fejlécek, törzs).

fetch Válasz Objektum

A `fetch` által visszaadott válasz objektum tartalmazza a kéréssel kapcsolatos információkat, például a státuszkódot és a válasz törzsét.

Válasz Objektum Példája:

```
fetch("https://api.example.com/data")
    .then((response) => {
        console.log(response.status); // Státuszkód
        console.log(response.statusText); // Státusz szöveg
        return response.json(); // Válasz törzsének JSON formátumra alakítása
    })
    .then((data) => {
        console.log(data);
    });
```

Összegzés

Hálózati Kérés (Network Request)

A hálózati kérések lehetővé teszik, hogy a kliens adatokhoz férjen hozzá egy szerveren keresztül. Az API-k és HTTP metódusok használatával különböző műveleteket hajthatunk végre, mint például adatok lekérése vagy küldése.

API (Application Programming Interface)

Az API-k lehetővé teszik különböző alkalmazások közötti kommunikációt és adatcserét. Az API-k meghatározzák a kommunikációs szabályokat és protokollokat.

HTTP Metódusok

A HTTP metódusok meghatározzák, hogy milyen műveleteket hajthatunk végre egy erőforráson:

- **GET**: Adatok lekérése.
- **POST**: Adatok küldése.
- **PUT**: Adatok módosítása vagy létrehozása.
- **DELETE**: Adatok törlése.
- **PATCH**: Adatok részleges módosítása.

JSON (JavaScript Object Notation)

A JSON egy könnyű adatcsere formátum, amelyet gyakran használnak az API-kban. Könnyen olvasható és elemezhető mind emberek, mind gépek számára.

fetch Használata

A `fetch` API segítségével aszinkron HTTP kéréseket küldhetünk és fogadhatunk adatokat a szerverről. A `fetch` használata egyszerű és modern módja a hálózati kérések kezelésének JavaScript-ben.

A fenti példák és magyarázatok segítségével könnyebben megérthetjük és alkalmazhatjuk a hálózati kéréseket, API-kat, HTTP metódusokat, JSON formátumot és a `fetch` API-t a JavaScript alkalmazásokban.