Asynchronous Programming

Asynchronous Programming (česky asynchronní programování) je programovací technika, při které program nemusí čekat na dokončení časově náročné operace a může mezitím pokračovat ve vykonávání dalšího kódu.

Tento přístup je běžný zejména ve webových aplikacích, síťové komunikaci, práci se soubory a serverových technologiích, jako jsou JavaScript: Kompletní monografie od A do Z a Node.js.

Proč vzniklo asynchronní programování

Některé operace mohou trvat relativně dlouho:

načítání dat ze serveru, komunikace s databází, čtení a zápis souborů, síťová komunikace, stahování obsahu z internetu.

Pokud by program na dokončení těchto operací čekal, mohl by být neaktivní nebo zablokovaný.

Synchronní vs. asynchronní programování

Synchronní zpracování

Při synchronním zpracování program čeká na dokončení každého kroku.

```
Krok 1 ↓ Dokončení ↓ Krok 2 ↓ Dokončení ↓ Krok 3
```

Výhodou je jednoduchost, nevýhodou možné blokování programu.

Asynchronní zpracování

Při asynchronním zpracování může program pokračovat v práci, zatímco se operace provádí na pozadí.

```
Spuštění operace ↓ Program pokračuje dál ↓ Operace dokončena ↓ Zpracování výsledku
```

Příklad v JavaScriptu

Synchronní kód:

```
console.log("První"); console.log("Druhý"); console.log("Třetí");
```

Výstup:

```
První Druhý Třetí
```

Asynchronní kód:

```
console.log("První");

setTimeout(() => {
console.log("Druhý");
}, 1000);

console.log("Třetí");
```

Výstup:

```
První Třetí Druhý
```

Callbacky

Historicky byly asynchronní operace řešeny pomocí Callback funkcí.

```
setTimeout(function() { console.log("Hotovo"); }, 1000);
```

Při větším množství callbacků mohl vzniknout problém označovaný jako `Callback Hell`.

Promise

Modernější řešení představují promise objekty.

```
fetch("/data") .then(response => response.json()) .then(data =>
console.log(data));
```

Promise reprezentuje hodnotu, která bude dostupná až v budoucnu.

Async/Await

Nejpřehlednější způsob zápisu asynchronního kódu nabízí konstrukce `async` a `await`.

```
async function nactiData() { const response = await fetch("/data"); const
data = await response.json();
console.log(data);

}
```

Kód se podobá synchronnímu zápisu, ale stále pracuje asynchronně.

Event Loop

V prostředí JavaScriptu je asynchronní programování úzce spojeno s mechanismem Event Loop.

Event Loop:

sleduje frontu událostí, zpracovává callbacky, koordinuje asynchronní operace, umožňuje efektivní využití jednoho vlákna.

Oblasti použití

Webové aplikace. REST API. Databázové operace. Streamování dat. Síťová komunikace. Mobilní aplikace. Cloudové služby. IoT systémy.

Výhody

Vyšší výkon aplikace. Lepší odezva uživatelského rozhraní. Efektivní využití systémových prostředků. Možnost současného zpracování více úloh.

Nevýhody

Složitější návrh aplikace. Náročnější ladění chyb. Obtížnější sledování toku programu. Riziko závodních podmínek (race conditions).

Související pojmy

JavaScript: Kompletní monografie od A do Z Node.js Callback promise async await Event Event Loop Event-Driven Programming multithreading

Shrnutí

Asynchronous Programming je způsob programování, při kterém aplikace nemusí čekat na dokončení časově náročných operací. Může pokračovat ve vykonávání dalších úloh a výsledek zpracovat až ve chvíli, kdy je připraven. Díky tomu jsou moderní aplikace rychlejší, lépe reagují na uživatele a efektivněji využívají systémové prostředky.