DR INŻ. PIOTR SIEWNIAK

JavaScript – kod synchroniczny i asynchroniczny

WPROWADZENIE

OPERACJE SYNCHRONICZNE

1. **JavaScript jest językiem programowania jednowątkowym** (*single threaded programming language*). Oznacza to, że w danej chwili może być wykonywana tylko jedna (pojedyncza) operacja. Kolejna, następna operacja w kodzie źródłowym może się rozpocząć dopiero po zakończeniu wykonywania poprzedniej operacji. Takie operacje nazywane są **operacjami synchronicznymi** (*synchronous operations*).
2. Operacje synchroniczne, czyli operacje wykonywane sekwencyjnie jedna po drugiej, stanowią tzw. **operacje blokujące** (*blocking operations*). Wykonanie następnej operacji synchronicznej jest możliwe dopiero po zakończeniu wykonania poprzedniej operacji synchronicznej.
3. Mechanizm wykonywania operacji (funkcji) synchronicznych opiera się na wykorzystaniu tzw. **stosu wywołań funkcji** (*function call stack*), nazywanego także inaczej **stosem wykonań funkcji** (*function execution stack*). Ze względu na fakt, że JavaScript jest językiem programowania jednowątkowym, ma ona do dyspozycji tylko jeden stos wywołań funkcji. Tym samym, w danej chwili może być wykonywana tylko jedna operacja.
4. Wywołanie funkcji w języku JavaScript powoduje dodanie jej adresu zwrotnego (return adress), parametrów i argumentów oraz innych zmiennych lokalnych, na stos wywołań funkcji – a dokładnie na wierzch tego stosu. Po dodaniu funkcji na wierzch stosu rozpoczyna się jej wykonywanie. Domyślnie, każdy wiersz zawarty w ciele funkcji jest wykonywany sekwencyjnie, po jednym wierszu na raz. Jeśli w danej funkcji wywoływana jest inna funkcja, ona również jest dodawana na stos wywołań funkcji – ponad jej funkcją wywołującą (*caller*). Mechanizm ten powtarza się zgodnie ze stopniem zagnieżdżenia wywołań funkcji w kodzie źródłowym. Po zakończeniu wykonywania danej funkcji jest ona usuwana ze stosu, potem usuwana jest kolejna funkcja itd. W danej chwili na wierzch stosu wywołań funkcji może zostać dodana kolejna wywoływana funkcja itd.

Podsumowując:

kod synchroniczny to kod, w którym linie (instrukcje) są wykonywane sekwencyjnie jedna po drugiej. Jeśli daną instrukcję stanowi wywołanie funkcji, program czeka aż skończy się jej działanie i sterowanie wróci na poziom aplikacji. Dopiero wtedy wykonywana jest następna instrukcja (np. kolejna pętla, funkcja) występująca w kodzie źródłowym po wywołaniu funkcji.

OPERACJE ASYNCHRONICZNE

1. W języku JavaScript można również programować tzw. **operacje asynchroniczne** (*asynchronous operations*), gdzie rozpoczęcie danej operacji – bez konieczności jej ukończenia – nie wyklucza rozpoczęcia realizacji innej operacji. Dlatego też, w programach asynchronicznych w tym samym czasie różne operacje mogą być realizowane niezależnie od innych.
2. W dużym uproszczeniu – programowanie asynchroniczne polega na tym, że wykonywanie operacji nie przebiega w takiej kolejności, w jakiej występuje w kodzie źródłowym, tj. sekwencyjnie „linia po linii”.
3. W programowaniu asynchronicznym program nie czeka aż określone zadanie asynchroniczne zostanie zrealizowane do końca (czyli operacja asynchroniczna dobiegnie końca) – wykonywanie pozostałych operacji synchronicznych może być (i jest) kontynuowane.

4. Patrząc z innej strony, kod asynchroniczny to taki kod, który zawiera implementację co najmniej jednego zadania asynchronicznego.

5. **Asynchroniczność w JavaScripcie można implementować przy wykorzystaniu:**
    a. **funkcji zwrotnych (*callback functions*);**
    b. **obietnic (*promises*);**
    c. **funkcji asynchronicznych async (wraz z użyciem operatora await).**

REALIZACJA OPERACJI ASYNCHRONICZNYCH PRZY UŻYCIU FUNKCJI ZWROTNYCH

1) Pomimo tego, że silnik V8 języka JavaScript jest jednowątkowy, interfejs (API) C++ Node.js nie jest jednowątkowy. Oznacza to, że wywołanie jakiejś operacji nieblokującej może być obsługiwane w innym wątku – równolegle do operacji na stosie wywołań funkcji.

2) W czasie wykonywania dowolnej funkcji asynchronicznej, funkcja zwrotna stanowiąca jej argument, jest wykonywana w sposób asynchroniczny – tj. dopiero po zakończeniu wykonywania jej funkcji nadrzędnej (*parent function*), ineczej: funkcji wywołującej (*caller*). Innymi słowy, funkcja zwrotna jest wykonywana dopiero po zakończeniu wykonywania skojarzonej z nią operacji asynchronicznej.

3) W celu obsługi kodu asynchronicznego JavaScript musi sterować nie tylko **stosem wywołań funkcji**, ale również:
    a) **kolejką wywołań funkcji zwrotnych** (*callback queue / task queue*);
    b) **pętlą zdarzeń** (*event loop*).

4) Funkcja zwrotna stanowiąca argument funkcji asynchronicznej nie jest umieszczana na stosie wywołań funkcji w sposób bezpośredni, tj. niezwłocznie po jej wywołaniu w ciele funkcji nadrzędnej (wywołującej). Po wywołaniu danej funkcji asynchronicznej, funkcja zwrotna będąca jej argumentem jest najpierw dodawana do tzw. kolejki (wywołań) funkcji zwrotnych. Funkcja ta oczekuje w kolejce aż do chwili, w której stos wywołań funkcji stanie się pusty. Jeśli stos wywołań funkcji jest pusty, wówczas funkcja zwrotna jest pobierana z kolejki i umieszczana na stosie. Działanie to jest realizowane za pośrednictwem tzw. pętli zdarzeń. Po uruchomieniu i wykonaniu funkcja zwrotna jest usuwana ze stosu.

5) Patrząc z punktu widzenia modelu programowania sterowanego zdarzeniami (*event–driven programming model*), JavaScript sprawdza (monitoruje) kolejkę funkcji zwrotnych cyklicznie – w pętli zdarzeń. Jeśli dane zdarzenie skojarzone z wywołaniem określonej funkcji zwrotnej zostanie przechwycone, wówczas funkcja ta jest pobierana z kolejki i umieszczana na stosie wywołań funkcji – oczywiście pod warunkiem, że stos jest pusty. Funkcja zwrotna umieszczona na stosie jest wykonywana podobnie do zwykłych funkcji synchronicznych i analogicznie jak one po zakończeniu jest usuwana ze stosu.

6) Dzięki wykonywaniu opisanych powyżej działań za pośrednictwem kolejki funkcji zwrotnych i pętli zdarzeń (czyli **poza silnikiem V8**) nie powstaje żaden mechanizm blokowania realizacji innych operacji, ani oczekiwania na ich wykonanie. Funkcja zwrotna stanowiąca argument funkcji asynchronicznej zostaje umieszczona na stosie wywołań funkcji dopiero wtedy, gdy jest on pusty – czyli wykonywanie innych funkcji oraz funkcji nadrzędnej zostało już zakończone.

7) Biorąc pod uwagę powyższe, funkcje asynchroniczne nazywane są funkcjami nieblokującymi (*non-blocking functions*), a operacje realizowane za pośrednictwem funkcji asynchronicznych – operacjami nieblokującymi.