NODE.js

TABLICE INFORMATYCZNE • Mariusz Walczak

Helion

Globalne zmienne

`__filename` — ścieżka bezwzględna aktualnie wykonywanego skryptu.

`__dirname` — nazwa katalogu, w którym znajduje się aktualnie wykonywany skrypt (ścieżka bezwzględna).

`console` — referencja do konsoli.

`module` — referencja bieżącego modułu.

`global` — referencja globalnej przestrzeni nazw.

`process` — obiekt globalny, reprezentujący proces.

`Buffer` — typ globalny do przechowywania danych binarnych.

Liczniki czasu

`setTimeout(callback, delay[, ...args])` — ustawia opóźnienie wykonywanego skryptu.

`clearTimeout(t)` — czyści ustawienia opóźnienia dla obiektu typu `Timeout`.

`setInterval(callback, delay[, ...args])` — ustawia interwał wykonywania danej czynności.

`clearInterval(t)` — czyści interwał ustawiony komendą `setInterval()`.

`setImmediate(callback[, ...args])` — wykonuje daną operację asynchronicznie.

`clearImmediate(immediateObject)` — anuluje zadanie stworzone przez `setImmediate()`.

Moduły

`const loadedModule = require('./exampleModule')` — wczytuje moduł `exampleModule`.

`const loadedModule = require('nazwaModulu')` — wczytuje moduł zainstalowany z npm lub wbudowany w node

`loadedModule.require('./exampleModule')` — wczytuje moduł `exampleModule`.

`loadedModule.id` — identyfikator modułu.

`loadedModule.filename` — nazwa pliku modułu.

`loadedModule.loaded` — wskazuje, czy moduł jest załadowany.

`loadedModule.parent` — referencja do modułu, który jako pierwszy zażądał modułu `module`

`loadedModule.children` — zwraca moduły, które zostały wczytane w zasięgu modułu `module`.

Util

`util.format(format[, ...args])` — zwraca sformatowany ciąg tekstowy, działa jak `printf`.

`util.inspect(object, [opts])` — zwraca reprezentacje obiektu w celu sprawdzenia go i debugowania.

URL

`url.parse(urlStr, [parseQueryString], [slashesDenoteHost])` — zwraca obiekt URL.

`url.format(urlObj)` — zwraca sparsowany obiekt URL jako ciąg tekstowy.

`url.resolve(from, to)` — rozwiązuje docelowy URL w stosunku do bazowego.

Testowanie

`assert (warunek[, informacja])` — sprawdza, czy dany warunek się zgadza, a jeśli tak, to zwraca informację; alias `assert.ok()`.

```
const spodziewanaWartosc = 3
assert(spodziewanaWartosc === 3, 'zgadza się')
```

`assert.equal(wartoscSprawdzana, wartoscOczekiwana[,informacja])` — sprawdza, czy wartość sprawdzana jest równa oczekiwanej, a jeśli tak, to zwraca informację.

`assert.notEqual(wartoscSprawdzana, wartoscOczekiwana [,informacja])` — sprawdza, czy wartość sprawdzana nie jest równa oczekiwanej, a jeśli tak, to zwraca informację.

`assert.deepEqual(wartoscSprawdzana, wartoscOczekiwana [,informacja])` — sprawdza dogłębnie, czy dane tablice lub obiekty są sobie równe.

```
const assert = require('assert')

const obj1 = {
  a : {
    b : 1
  }
}
const obj2 = {
  a : {
    b : 2
  }
}
const obj3 = {
  a : {
    b : 1
  }
}
const obj4 = Object.create(obj1)

assert.deepEqual(obj1, obj1)
// OK, wykaże, że obiekt jest identyczny

assert.deepEqual(obj1, obj2)
// AssertionError: { a: { b: 1 } } deepEqual { a: { b: 2 } }
// wartości a i b są różne

assert.deepEqual(obj1, obj3)
// OK, wykaże, że obiekt jest identyczny

assert.deepEqual(obj1, obj4)
// AssertionError: { a: { b: 1 } } deepEqual {}
```

`assert.notDeepEqual(wartoscSprawdzana, wartoscOczekiwana [,informacja])` — sprawdza dogłębnie, czy dane tablice lub obiekty nie są sobie równe; najlepiej używać do obiektów i tablic.

`assert.strictEqual(wartoscSprawdzana, wartoscOczekiwana [,informacja])` — sprawdza, czy wartość sprawdzana jest identyczna z oczekiwaną (wartość i typ muszą się zgadzać), a jeśli tak, to zwraca informację; najlepiej używać do zmiennych.

`assert.notStrictEqual(wartoscSprawdzana, wartoscOczekiwana [,informacja])` — sprawdza, czy wartość sprawdzana jest identyczna z oczekiwaną (wartość i typ muszą się zgadzać), a jeśli tak, to zwraca informację; najlepiej używać do zmiennych.

`assert.deepStrictEqual(wartoscSprawdzana, wartoscOczekiwana [,informacja])` — sprawdza dogłębnie, czy dane tablice lub obiekty są identyczne (czyli muszą być równe co do wartości i typu); najlepiej używać do obiektów i tablic.

`assert.notDeepStrictEqual(wartoscSprawdzana, wartoscOczekiwana [,informacja])` — sprawdza dogłębnie, czy dane tablice lub obiekty nie są identyczne (czyli nie mogą być równe co do wartości i typu); najlepiej używać do obiektów i tablic.

`assert.throws(blok[,błąd, informacja])` — sprawdza, czy podana funkcja zwraca błąd. Opcjonalnie możemy określić rodzaj błędu, podając go jako drugi parametr. Po spełnieniu warunku zostanie wyświetlony komunikat zawarty w parametrze informacja.

```
assert.throws(
  () => {
    throw new Error('Zła wartość')
  },
  Error
)
```

Bufor (Buffer)

`Buffer` został stworzony do działania na danych binarnych, jest oparty na standardzie `TypedArrays`, który opisuje tablice liczb całkowitych.
Bufora używa się do obsługi strumienia danych.

`Buffer.alloc(rozmiar[, wypelnienie[, kodowanie]])` — tworzy nowy bufor o danym rozmiarze, wypełnia go danymi — `wypelnienie` — i konwertuje go z konkretnego kodowania na tablicę liczb całkowitych.

```
const buf = Buffer.alloc(5)

// Zwraca: <Buffer 00 00 00 00 00>
console.log(buf)

const buf = Buffer.alloc(5, 'a')

// Zwraca: <Buffer 61 61 61 61 61>
console.log(buf)

const buf = Buffer.alloc(11, 'aGVsbG8gd29ybGQ=', 'base64')

// Zwraca: <Buffer 68 65 6c 6c 6f 20 77 6f 72 6c 64>
console.log(buf)
```

`Buffer.byteLength(ciągZnaków[, kodowanie])` — zwraca długość bufora wypełnionego danym ciągiem znaków, przekonwertowanego z danego kodowania.

`Buffer.compare(buffer1, buffer2)` — porównuje, czy dwa bufory są sobie równe.

```
const buf1 = Buffer.from('1234')
const buf2 = Buffer.from('0123')
const arr = [buf1, buf2]

// Zwraca: [ <Buffer 30 31 32 33>, <Buffer 31 32 33 34> ]
```