Език за програмиране JavaScript

Web технологии (ИС)
ФМИ, СУ

История на езика

- Създаден през 1995 г. от Brendan Eich
  - ... за около седмица
- Името е маркетингов трик, без пряка връзка с Java
- Използван в над 97% от уеб сайтовете

източник: https://www.reddit.com/r/ProgrammerHumor/comments/621qrt/javascript_the_good_parts/

Основни характеристики

- Слабо типизиран
  - обектите не са обвързани с конкретен тип
- Динамичен
  - типът може да се променя по време на изпълнение
- Прототипно базирано ООП
  - типовете образуват “верига” от прототипи
  - ... подобно на Python
- Функциите са “първокласни” типове
  - могат да се създават динамично
  - съхраняват в променливи
  - предават като аргументи на други функции
  - синтактично погледнато: равнопоставени с останалите вградени типове

Повсеместност

- В трите слоя на едно типично уеб приложение
  - клиентска част (front end), изпълнява се в браузъра
  - сървърна част (back end), среди (runtimes) като Node.js
  - в някои бази данни като MongoDB

Променливи и типове

Различни обхвати на променливите (scope)

- Глобален - общ за цялата програма
  - без ключова дума
- let/const
  - на ниво блок (което е != функция)
  - декларирането се изпълнява преди останалия код (hoisting)
- var
  - обхват на ниво функция или глобален
  - по подразбиране се инициализира на undefined
- със или без задаване на стойност

```
let name = 'Генчо';
var nonexistent;  // undefined
```

Примери за hoisting

- на променливи

```
console.log(not_initialized); // undefined
var not_initialized;

console.log(not_defined); // хвърля изключение
                          // ReferenceError: not_defined is not defined
let not_defined;
```

- на функции

```
foo(42);        // 42 <-- foo е достъпна преди декларацията си
function foo(n) {
  console.log(n);
}
```

Примитивни типове

- Най-популярните
  - числа (с плаваща запетая) - Number
  - символни низове - String
  - булеви стойности - true|false
  - undefined - стойност по подразбиране за току-що декларирани променливи и аргументи на функция
- Непроменяеми (immutable)
- Нямат собствени свойства (освен вградени)
- Други
  - null
  - дълги числа - BigInt
  - символи - Symbol

Структурни типове

- Обекти
  - Object, {}
  - колекции от свойства, съответстващи на даден ключ - properties
  - ключовете са низове или символи
  - стойностите им могат да бъдат всички типове
  - всеки обект има прототип, който може да бъде друг обект

- Масиви
  - Array, []
  - също са обекти
  - имат вградено, непроменяемо свойство length

Други колекции

- Set
  - множество от стойности, без повторения (в смисъл на “===”)
- Map
  - съответсвие ключ -> стойност
  - без повторения в ключовете
- WeakSet / WeakMap
  - не може да се обхождат ключовете
  - ... но пестят памет (в общия случай)

Управляващи конструкции

- Цикли
  - класически for със C-подобен синтаксис
  - for...in - обхожда имената на свойствата на обекти
  - for...of - обхожда елементите на масиви, низове, Map, Set, arguments
  - while, do...while

- Условия
  - if...else
  - switch - сравнява изрази чрез стриктно сравнение, “===”

Функции

- Функциите също са обекти
  - от вградения тип Function
- Вградени свойства:
  - име
  - референция към контекст на изпълнение (this)
  - length - дължина на списъка с аргументи
- Методи за “свързване” с контекст
  - bind, apply, call
  - връщат нова функция
  - this придобива стойност на конкретен обект

Пример - FizzBuzz

```
function fizzbuzz(n) {
  for (let i = 1; i < n; i++) {
    let out = '';
    if (i % 3 == 0) out += 'fizz';
    if (i % 5 == 0) out += 'buzz';
    if (!out) out = i.toString();
    console.log(out);
  }
}
```

ООП

Създаване на обекти

- Чрез директно инициализиране (object literal)

```
const Animal = {};
```

- Чрез конструктора на класа Object

```
const Animal = new Object();
```

Видове свойства на обектите

- Вградени - дефинирани от езика

```
Animal.constructor === Object;  // true
```

- Собствени - дефинирани върху самия обект

```
Object.defineProperty(Animal, 'name',
{
   writable: true,
   enumerable: true
});
Object.hasOwn(Animal, 'name');  // true
```

- Наследени - дефинирани върху някой от прототипите

Прототипи на обектите

- Всеки обект има прототип, от който се създава

```
let my_cat = Object.create(Animal);
```

  - може да се чете динамично

```
Object.getPrototypeOf(my_cat) === Animal; // true
```

  - може и да се променя
  - ... но това забавя работата на изпълнение на цялата програма

- Прототипът е шаблон със свойства, общи за създадените от него обекти

```
my_cat.hasOwnProperty('name');   // true
my_cat.name = 'Панка';           // my_cat['name'] === 'Панка'
```

Функции и свързване с контекст

- Function.prototype.bind, .apply

```
function greet() {
  if (this.name)
    console.log('Здравей, ', this.name);
  else
    console.log('Здравей, кой си ти?');
}

greet();          // Здравей, кой си ти?

const cat_greeter = greet.bind(my_cat);
cat_greeter();    // Здравей, Панка
```

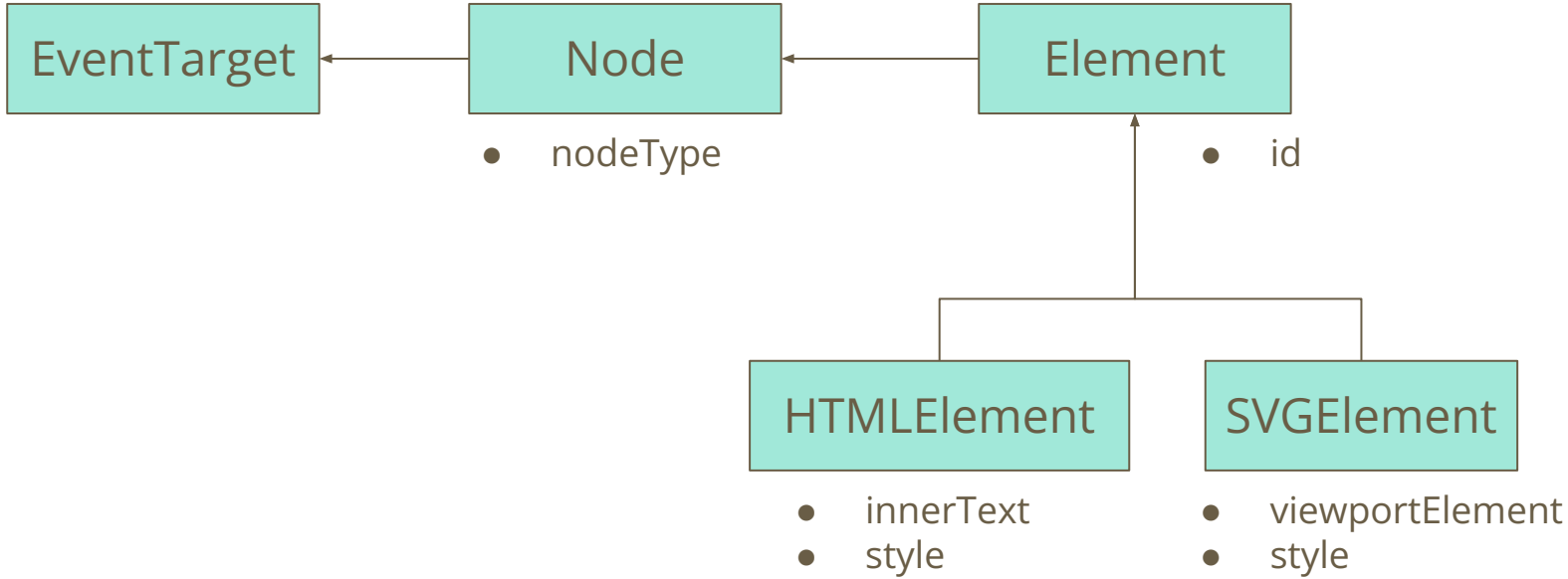
Йерархия от типове

- Релацията “прототип на” изгражда вериги (едносвързани списъци)
- Веригите се разклоняват в дървета
- Всяко от дърветата има за корен Object

Например в HTML:

```
Object.getPrototypeOf(HTMLElement) === Element;     // true
Object.getPrototypeOf(HTMLElement)
    === Object.getPrototypeOf(SVGElement);          // true
```

Пример с дефинирани от HTML типове



Изчислителен модел, асинхронност

Event loop

- Event loop - основа на средата за изпълнение на JS
  - безкраен цикъл, който изпълнява части от кода на приложението
  - приложението добавя събития в опашката - напр. чрез setTimeout
  - средата изтегля от опашката събития за обработка
  - ... и изпълнява задачите (callbacks), регистрирани към дадено събитие

- Сходни концепции
  - честа практика в построяването на игри
  - message loop в графичните приложения за Windows

Отложено изпълнение на задачи (callbacks)

- `setTimeout ( callback, delayMs )`
  - изпълнява callback след delayMs милисекунди
- `setInterval ( callback, intervalMs )`
  - изпълнява callback през intervalMs милисекунди
  - връща число, идентификатор на отложената задача (id)
- `clearTimeout ( id ) | clearInterval ( id )`
  - прекъсват отложената задача

*) имплементирани от средата, не са част от езика

Пример за периодични задачи - таймер

```
const timer1s = setInterval(function timer() {
  console.log('tick');
}, 1000);

// tick
// tick
// [...]
clearInterval(timer1s);
```

Arrow function expressions

- Не създават собствен контекст
  - заемат този на обхващащата ги функция (lexical scoping)
  - сходни с lambda expressions в други езици

- В резултат:
  - нямат собствени this, super
  - не могат да се ползват като методи
  - не могат да се ползват като конструктори
  - не могат да се ползват като цел за bind, call, apply

```
setInterval(() => console.log('tick'), 1000);
```

Closures

- Closure - функция в комбинация със заобикалящата я среда (scope)
- Позволява да използваме референции към променливи от външната среда

```
function timer() {
  let n = 0;
  return function tick() {
    console.log(n++);
  }
}
```

Пример - хронометър

```
const s = new stopwatch(1000);
s.start();

// 0
// 1
// 2
// [...]
s.stop();
```

```
function stopwatch(intervalMs) {
  let n = 0;
  let timerId;

  this.start = function start() {
    timerId = setInterval(
      () => console.log(n++),
      intervalMs);
  }
  this.stop = function stop() {
    clearInterval(timerId);
    n = 0;
  }
  return this;
}
```

Съвременни средства за писане на асинхронен код

- Promises - подобно на std::future в C++
- генератори - function*, yield
- асинхронни функции - async/await

Пример за ползване на асинхронни APIs

Fetch API

```
fetch(url)
  .then(response => {
    // проверка за типа на response
    return response.json();
  })
  .then(data => {
    console.log(data);
  })
  .catch(error => console.error(error));
```

Благодаря!

Коментари, въпроси..?

в Moodle: Мая Лекова

https://twitter.com/zmayski

Дискусия от лекцията

- null vs. undefined
  - undefined е (глобална) константа за маркиране на недефинирани променливи
  - undefined се връща от функции, които нямат return
  - null отбелязва, че променливата е дефинирана, но (в този момент) няма стойност
  - и двете са примитивни
- оператори за сравнение
  - стриктното сравнение “===” счита стойности от различен тип като различни
  - сравнението “==” се опитва да конвертира типовете преди сравнение
  - … което понякога не е това, което очаквате
- ефективна манипулация на символни низове
  - с вградени методи на езика
  - чрез Encoding API и типове DataView, ArrayBuffer за работа със сурови данни (пример)

Допълнителни материали

- Всички самоучители в MDN
- Книга “Eloquent JavaScript” (достъпна директно онлайн)
- Event loop нагледно
  - “What the heck is the event loop anyway?” - чудесно видео (26 m) по темата
  - “In The Loop” - още едно видео (35 m) как браузърът си взаимодейства с JS, за да показва/обновява сайта на екрана
- Нови функционалности в езика, през погледа на V8

Материали в помощ на учебните презентации

- First Class Citizens
  - Функциите като "First class citizens" в езиците за програмиране
  - Closures в JavaScript
  - Библиотеки за функционално програмиране на JS
- Prototype-based OOP
  - Обзор на Прототипно-базираното ООП и сравнение с "традиционното" ООП
  - Примери в JS
  - За какво служи запазената дума this в JS?
- Data Types
  - а) Вътрешно представяне на основните типове в JS. Какво представляват масиви, обекти, числа, низове и др.
  - б) Итератори и генератори на JS. Смисъл и примери.