УДК 681.518

АНАЛІЗ КЛАСІВ МОВИ ПРОГРАМУВАННЯ JAVASCRIPT

Щербаков Є.В., Щербакова М.Є.

ANALISIS OF PROGRAMMING LANGUAGE JAVASCRIPT CLASSES

Shcherbakov E.V., Shcherbakova M.E.

Коротко подані та проаналізовані основні можливості об'єктно-орієнтованого програмування (ООП) на основі класів, введених в мову програмування JavaScript специфікацією ECMAScript 2015 [1]. Зокрема, розглянуті, досліджені та запропоновані варіанти оптимального використання в програмах на JavaScript основних парадигм ООП: властивостей та методів класів, спадкування класів, інкапсуляції та поліморфізму. Зроблено висновок, що головна мета введення класів - це простий та зрозумілий синтаксис для роботи з функціями-конструкторами та прототипним спадкуванням JavaScript.

Ключові слова: *JavaScript, об'єктно-орієнтоване програмування, класи, інкапсуляція, спадкування, поліморфізм*

Вступ. Об'єктно-орієнтоване програмування в сучасних мовах програмування є розвитком двох концепцій: концепції модульного програмування, що з'явилася в більш ранніх мовах програмування, таких як С, та концепції типів даних користувача, представлених структурами в тій же мові С, що поєднують в собі різнотипні складові елементи. Так само, як і при модульному програмуванні, при об'єктно-орієнтованому програмуванні дані та функції, що їх оброблюють, об'єднуються разом, утворюючи клас. Але на відміну від програмного модуля клас не виконує ніяких реальних дій ні самостійно, ні за запитом від інших частин програми. Клас є штампом, за допомогою якого виробляються однотипні функціональні модулі програми, в сукупності звані об'єктами. Дані та функції кожного об'єкту тісно зв'язані один з одним. Саме об'єкти виконують всю реальну роботу при виконанні об'єктно-орієнтованих програм.

До 2015 року синтаксис класів в мові програмування JavaScript був відсутній. Концепція класів була введена в цю мову специфікацією ECMAScript 15 та носить назву «максимально мінімальної». Тобто, в неї увійшли тільки ті можливості, які вже зовсім необхідні. Зокрема, не ввійшли «приватні» і «захищені» властивості, тобто, всі властивості класу доступні зовні.

Основна частина. До появи специфікації ECMAScript 15 [1] в мові JavaScript [2] активно використовувались об'єкти і свій особливий тип спадкування на базі прототипів. Об'єкти в JavaScript могли мати довільне число властивостей і ці властивості могли додаватися в об'єкт динамічно. В строго типізованих мовах, таких як C++ та Java, це не так. У них будь-який об'єкт має заздалегідь визначений набір властивостей, а кожна властивість має строго визначений тип. У C++ та Java клас визначає структуру об'єкта. Клас задає поля, які містяться в об'єкті, і типи даних цих полів. Він також визначає методи для роботи з об'єктом [3].

Специфікація ECMAScript 15, на додаток до вже наявних можливостей об'єктно-орієнтованого програмування, вводить в мову JavaScript традиційний синтаксис класів з об'єктними та статичними властивостями, а також класичним спадкуванням. Проаналізуємо основні можливості об'єктно-орієнтованого програмування сучасної мови JavaScript на базі класів більш детально.

Методи класів. В тілі класу JavaScript дозволяється визначати тільки властивості-методи і не дозволяються властивості-значення. Якщо властивість-значення все ж потрібна, то треба створювати одержувач та/або установник властивості, який буде повертати та/або встановлювати потрібне значення.

Методи, які можуть бути членами класу, можна розділити на три групи: конструктор класу, прототипні та статичні методи класу.

Як і в класичних мовах програмування, завдання конструктора класу - створити та ініціалізувати об'єкт класу, коли конструктор викликається на виконання за допомогою оператора new.

Прототипні методи класу формуються як властивості об'єкта ім'я_класу.prototype і, таким чином, стають методами, які викликаються на виконання через об'єкти даного класу, тобто об'єктними методами класу.

Статичні методи визначаються за допомогою ключового слова static, стають методами

конструктора класу і таким чином можуть викликатись на виконання, використовуючи тільки ім'я класу, як і в інших мовах програмування.

На відміну від класичних мов програмування, ім'я методу класу JavaScript можна визначити за допомогою виразу, якщо помістити цей вираз в квадратні дужки. Наприклад, такі три варіанти визначення класу CmClass еквівалентні:

```
class CmClass {
        myMethod() {}
}
class CmClass {
        ['my'+'Method']() {}
}
let m = "myMethod";
class CmClass {
        [m]() {}
}
```

Всі методи класів JavaScript є непереліковними, тобто вони не видимі при обробці властивостей об'єктів класів за допомогою циклів for/in.

Інкапсуляція. Одна з найбільш загальних характеристик класичних об'єктно-орієнтованих мов програмування, таких як C++ та Java, полягає в можливості оголошення приватних (private) властивостей класу, звертатися до яких можна тільки з методів цього класу і недоступних за межами класу. Поширена техніка програмування, звана інкапсуляцією даних, полягає в створенні приватних властивостей і організації доступу до цих властивостей тільки через спеціальні методи читання/запису.

Як було сказано раніше, в відповідності зі специфікацією ECMAScript 15, всі властивості класу доступні зовні. Але JavaScript дозволяє імітувати інкапсуляцію даних за допомогою замикань функцій, для чого необхідно, щоб методи доступу зберігалися в кожному об'єкті класу і з цієї причини не могли успадковуватися від об'єкта-прототипу. Наступний фрагмент демонструє, як можна домогтися цього. Він містить реалізацію класу прямокутників Rectangle, ширина і висота яких доступні і можуть змінюватися тільки шляхом звернення до спеціальних методів:

```
function ImmutableRectangle(w, h) {
this.getWidth = function() { return w; }
this.getHeight = function() { return h; }
}
// Клас може мати звичайні методи
// в об'єкті-прототипі
ImmutableRectangle.prototype.area = function() {
return this.getWidth() * this.getHeight();
};
```

Окрім замикань, для зберігання приватних даних класів в сучасній мові JavaScript можна використовувати слабкі мапи (weak maps), а також застосовувати унікальні ідентифікатори типу Symbol в якості ключів для приватних властивостей.

Спадкування. В відповідності до специфікації ECMAScript 15 в сучасній мові JavaScript можна створити новий клас, похідний від уже існуючого, вказавши ім'я базового класу після ключового слова extends. Базовий клас в цьому випадку називають суперкласом або батьківським класом, а похідний клас — субкласом або дочірнім класом. В дочірньому класі може використовуватись ключове слово super в спеціальному методі класу constructor() для виклику конструктора батьківського класу або в методах класу для посилань на прототипні та статичні методи батьківського класу або функції-конструктора. Якщо дочірній клас не має метода constructor(), за замовчуванням буде викликатись конструктор батьківського класу. Все це практично повторює синтаксичні механізми спадкування класів в класичних об'єктно-орієнтованих мовах програмування, таких як Java та їй подібних.

На відміну від C++ мова JavaScript підтримує тільки одиночне спадкування, оскільки при спадкуванні в більшості випадків створюваному класу достатньо одного суперкласу. Але це становиться обмеженням, коли створюються розгалужені ієрархії класів, а також коли при спадкуванні потрібно наслідувати інструментальні методи з різних джерел. Скажімо, в сценарії є клас Person і створюється підклас Employee:

```
class Person {}
class Employee extends Person {}
```

і при цьому потрібно наслідувати функції від двох службових класів: BackgroundCheck – цей клас робить перевірку даних співробітника, і Onboard - цей клас супроводжує прийом співробітника на роботу, наприклад, друкує бейджик і таке інше:

```
class BackgroundCheck {
    check() {}
}
class Onboard {
    printBadge() { }
}
```

Обидва класи, BackgroundCheck та Onboard, являють собою абстрактні підкласи, функціональні можливості яких можуть використовуватися багато разів, і які називаються міксинами (mixins).

Оскільки множинне спадкування в JavaScript відсутнє (в відповідності до ECMAScript 15), для досягнення початкової мети використовується інша техніка. Популярний спосіб реалізації міксинів в JavaScript - це написати функцію, в якості вхідного аргументу якій передається суперклас, і яка в якості вихідного значення повертає похідний від нього клас (підклас), наприклад:

```
class Person {}
    const BackgroundCheck = Tools => class
extends Tools {
    check() {}
};
const Onboard = Tools => class extends Tools {
    printBadge() {}
};
class Employee extends
BackgroundCheck(Onboard(Person)){
}
```

По суті це означає, що Employee є підкласом BackgroundCheck, який, в свою чергу, є підкласом Onboard, який, в свою чергу, є підкласом Person.

Хоча синтаксично спадкування класів в JavaScript дуже близьке до спадкування в класичних мовах програмування, воно насправді є тільки обгорткою над традиційним для JavaScript прототипним спадкуванням [4, 5], тобто є динамічним, а не статичним.

При використанні прототипного спадкування в JavaScript програма під час виконання може будь-коли додати, відкоригувати або видалити одну або декілька властивостей об’єкта на будь-якому рівні ієрархії спадкування, тим самим змінивши поведінку всього піддерева дочірніх об’єктів. При класичному спадкуванні все, що оголошено в кожному класі ієрархії спадкування залишається там назавжди, тобто класичне спадкування передбачає наявність певної статичної схеми, по якій будується кожен об'єкт дерева спадкування класів. Таким чином, при використанні прототипного спадкування ми маємо справу не зі схемою, а з живим організмом, який постійно розвивається, з часом змінюється і приймає ту форму, яка потрібна веб-сторінці, додатку або сервісу в поточний момент часу.

Поліморфізм. Поліморфізм - це можливість використання одного і того ж об’єктного метода для виконання різних дій. У мовах C ++ і Java дія, яка виконується в даний момент, визначається типом об'єкта, від імені якого був викликаний об’єктний метод. Поліморфізм в C ++ реалізується за допомогою віртуальних функцій, які викликаються через об'єктні змінні, що посилаються за допомогою вказівників на базовий клас [3].

В JavaScript поліморфна поведінка методів класів підтримується механізмом пошуку їх реалізацій по ланцюжку прототипів об’єктів, а також тим, що із-за слабкої типізації всі об'єктні змінні можна розглядати як вказівники на базовий клас Object [2]. Тому для поліморфної поведінки методів JavaScript досить показати можливість перевизначення методів базових класів в похідних класах.

Дійсно, коли на JavaScript в похідному класі визначається метод, який має те ж саме ім'я, що й метод базового класу, похідний клас перевизначає (overrides) цей метод. Найчастіше перевизначення методів проводиться не з метою повної заміни, а лише для того, щоб розширити їх функціональність. Для цього метод повинен мати можливість викликати перевизначений метод. У певному сенсі такий прийом за аналогією з конструкторами можна назвати викликом методів по ланцюжку. Однак викликати перевизначений метод набагато менш зручно, ніж викликати конструктор базового класу. Для демонстрації цього розглянемо наступний приклад. Припустимо, що клас прямокутників Rectangle визначає метод toString() наступним чином:

```
Rectangle.prototype.toString = function( ) {
return "[" + this.width + ", " + this.height + "]";
}
```

Цей метод необхідно перевизначити в похідному класі, наприклад, PositionedRectangle, щоб об'єкти похідного класу могли мати строкове представлення, яке відображає значення не тільки ширини і висоти, але і всіх інших властивостей прямокутника. З метою досягнення більшого узагальнення будемо обробляти значення властивостей координат в самому класі, а обробку властивостей width і height делегуємо базовому класу. Зробити це можна приблизно так:

```
PositionedRectangle.prototype.toString = function() {
return "(" + this.x + "," + this.y + ") " +
// поля цього класу
Rectangle.prototype.toString.apply(this);
// виклик методу суперкласу
// по ланцюжку прототипів
}
```

Реалізація методу toString() базового класу доступна як властивість об'єкта-прототипу базового класу. Не можна викликати метод безпосередньо - довелося скористатися методом apply(), щоб вказати, для якого об'єкта викликається метод. Однак якщо в PositionedRectangle.prototype додати властивість superclass, можна зробити так, щоб цей код не залежав від типу базового класу:

```
PositionedRectangle.prototype.toString = function( ) {
return "(" + this.x + ", " + this.y + ") " +
// поля цього класу
this.superclass.prototype.toString.apply(this);
}
```

Властивість superclass може використовуватися в ієрархії спадкування тільки один раз. Якщо воно буде використовуватись якимось класом і похідним від нього класом, це призведе до безкінечної рекурсії.

Висновки. Програмування на основі класів, введене в JavaScript специфікацією ECMAScript 15, - це тільки новий синтаксис використання існуючої об’єктно-орієнтованої моделі програмування.

Головною метою введення класів є простий та зрозумілий синтаксис для роботи з конструкторами та спадкуванням за допомогою прототипів.

По суті, класи є функціями. Вони лише надають новий синтаксис створення функцій, які застосовуються в якості конструкторів. Створення за допомогою класів функцій, які не використовуються як конструктори, не має будь-якого сенсу, не надає ніяких переваг, але робить код більш заплутаним та важким для читання. Тому класи слід використовувати тільки для створення об’єктів.

Хоча синтаксично спадкування класів в JavaScript дуже близьке до спадкування в класичних мовах програмування, воно насправді є тільки обгорткою над прототипним спадкуванням, тобто є динамічним, а не статичним.

При використанні прототипного спадкування в JavaScript програма під час виконання може будь-коли додати, відкоригувати або видалити одну або декілька властивостей об’єкта на будь-якому рівні ієрархії спадкування, тим самим змінивши поведінку всього піддерева дочірніх об’єктів.

При класичному спадкуванні все, що оголошено в кожному класі ієрархії спадкування залишається там назавжди, тобто класичне спадкування передбачає наявність певної статичної схеми, по якій будуються всі об'єкти дерева спадкування класів.

Для приховування (інкапсуляції) даних в об’єктах класів, формальний синтаксис для чого в специфікації ECMAScript 15 відсутній, можна користатися механізмом замикання функцій, слабкими мапами (weak maps) або унікальними ідентифікаторами Symbol.

Поліморфна поведінка методів класів в JavaScript підтримується механізмом пошуку їх реалізацій по ланцюжку прототипів, а також тим, що із-за слабкої типізації всі об'єктні змінні можна розглядати як вказівники на базовий клас Object.

Література

1. ECMAScript® 2015 Language Specification [Електронний ресурс]. - Режим доступу: http://www.ecma-international.org/ecma-262/6.0/index.html#sec-ecmascript-function-objects.
2. Рязанцев О.І. Мовні засоби розробки веб-додатків / Рязанцев О.І., Щербаков Є.В., Щербакова М.Є. ; СНУ ім. В. Даля. - Сєвєродонецьк: Вид-во СНУ ім. В. Даля, 2017. - 466 с.
3. Щербаков Є.В. Мовні засоби системного програмування: Навчальний посібник / Щербаков Є.В., Щербакова М.Є. ; СНУ ім. В. Даля. – Луганськ: Вид-во СНУ ім. В. Даля, 2006. – 376 с.
4. Ртищев В. ES6 классы [Електронний ресурс]. - Режим доступу: http://jsraccoon.ru/es6-classes.
5. Щербаков Е. В. Особенности объектно-ориентированного программирования в языке JavaScript / Е. В. Щербаков, М. Е. Щербакова // Вісник Східноукраїнського національного університету імені Володимира Даля. - 2011 - №10 (164), ч .2. - С. 229-234.

References

1. ECMAScript® 2015 Language Specification [Electronic resource]. - Access mode: http://www.ecma-international.org/ecma-262/6.0/index.html#sec-ecmascript-function-objects.
2. Ryazantsev O. I. Language means of web-application developing / Ryazantsev O.I., Shcherbakov E.V., Shcherbakova M.E.; EUNU. – Severodonetsk: Vyd-vo SNU im. V. Dalya, 2017. - 466 p.
3. Shcherbakov E.V. Language means of system programming: Tutorial / Shcherbakov E.V., Shcherbakova M.E. / EUNU. – Luhansk: Vyd-vo SNU im. V. Dalya, 2006. – 376 p.
4. Rtyshchev V. ES6 classes [Electronic resource]. - Access mode: http://jsraccoon.ru/es6-classes.
5. Shcherbakov E. V. Features of object-oriented programming in JavaScript language / E. V. Shcherbakov, M. E. Shcherbakova // Visnik of the Volodymyr Dahl East Ukrainian National University. - 2011 - №10 (164), part 2. - p. 229-234.

Щербаков Е.В., Щербакова М.Е. Анализ классов языка программирования JavaScript

Коротко представлены и проанализированы основные возможности объектно-ориентированного программирования (ООП) на основе классов, введенных в язык программирования JavaScript спецификацией ECMAScript 2015. В частности, рассмотрены, изучены и предложены варианты оптимального использования в программах на JavaScript основных парадигм ООП: свойств и методов классов, наследования классов, инкапсуляции и полиморфизма. Сделан вывод, что главная цель введения классов - это простой и понятный синтаксис для работы с функциями-конструкторами и прототипным наследованием JavaScript.

Ключевые слова: *JavaScript, объектно-ориентированное программирование, классы, инкапсуляция, наследование, полиморфизм*

Shcherbakov E.V., Shcherbakova M.E. Analysis of programming language JavaScript classes

Briefly presented and analyzed the main features of object-oriented programming (OOP) based on classes, introduced into the programming language JavaScript by specification ECMAScript 2015. In particular, the variants of the optimal use of basic OOP paradigms in JavaScript programs are discussed, studied and proposed: class properties and methods, class inheritance, encapsulation and polymorphism. It is concluded that the main purpose of introducing classes is a simple and understandable syntax for working with constructor functions and prototypical JavaScript inheritance.

Keywords: *JavaScript, object-oriented programming, classes, encapsulation, inheritance, polymorphism*

Щербаков Є.В. – к.т.н., доцент, доцент кафедри комп'ютерної інженерії Східноукраїнського національного університету ім. В. Даля, e-mail: gkvarc@gmail.com

Щербакова М. Є. – к.т.н., доцент, доцент кафедри комп'ютерної інженерії Східноукраїнського національного університету ім. В. Даля, e-mail: m.shcherbakova432@gmail.com

Рецензент: д.т.н., проф. ***Смолій В.М.***

Стаття подана 26.08.2017