УДК 004.9

ДОСЛІДЖЕННЯ JS КОМПІЛЯТОРІВ НА РІЗНИХ КЛАСАХ ЗАДАЧ

Носко Юлія

Науковий керівник: канд. тех. наук, доцент Баранюк О.Ф.

Центральноукраїнський державний педагогічний університет імені Володимира Винниченка, м. Кропивницький, Україна

В статті розглядаються компілятори мови JavaScript, та на основі істотних характеристик проводиться їх порівняння. Для різних класів задач досліджують різні види компіляторів та аналізується ефективність і застосування. Також у статті подається інформація про схеми роботи компіляторів, їх застосування в сучасному світі та еволюція, аналізується складність мови JavaScript та вплив цього на складність компіляторів. Проаналізована інформація дає підстави для висновків про те, чому схеми роботи компіляторів не є стандартними та чому існує велика кількість реалізацій компіляторів для мови JavaSrcript.

Ключові слова: js, компілятори, інтерпретатор, браузер, V8, Rhino, web, engine.

Research of usage JS compilers on different task classes

J. Nosko

Scientific supervisor: Candidate of Technical Sciences, Docent

Baranyuk A.F.

The article shows the information about JavaScript compilers, their comparison, on the basis of their essential characteristics. Different types of compilers are investigated for different task classes and their efficiency is analyzed. Also, the article provides information about the schemes of compilers, their usage in the modern world and evolution, analyzes the complexity of the JavaScript language and the impact of it on compilers complexity. The analyzed information gives conclusions about why the compilers work patterns are not standard and why there are a large number of implementations of compilers for the JavaSrcript language.

Key words: js, compiler, interpreter, browser, V8, Rhino, web, engine .

Постановка проблеми. Компіляція – це коли вихідний код програми, за допомогою спеціального інструменту, іншої програми, яка називається «Компілятор», перетворюється в іншу мову, як правило – в машинний код. Цей машинний код потім поширюється і запускається. При цьому вихідний код програми залишається у розробника.

Інтерпретація – це коли вихідний код програми отримує інший

інструмент, який називають «інтерпретатор», і виконує його. При цьому поширюється саме сам вихідний код (скрипт). Цей підхід застосовується в браузерах для JavaScript..

Сучасні інтерпретатори перед виконанням перетворюють JavaScript в машинний код або подібний до нього, оптимізують, а вже потім виконують. І навіть під час виконання намагаються оптимізувати. Тому JavaScript працює дуже швидко.

У всі основні браузери вбудований інтерпретатор JavaScript, саме тому вони можуть виконувати скрипти на сторінці. Але, зрозуміло, JavaScript можна використовувати не тільки в браузері. Це повноцінна мова, програми на якому можна запускати і на сервері.

Аналіз досліджень та публікацій. Зараз сприймається як належне швидке виконання js-коду в браузерах, і з кожним днем з'являється все більше прикладів того, що можна реалізувати за допомогою JS.

Першим engine, що інтерпретував js-код став SpiderMonkey, який був представлений в браузері Netscape 2.0 у 1995 р. У Брендана Айка було всього 10 днів на дизайн мови і побудову компілятора. Javascript був успішний з самого початку, і до серпня того ж року Майкрософт вже вбудувала свою версію JScript в Internet Explorer 3.0. До кінця 1996 мова був прийнятий комісією для формальної стандартизації, і вже в червні наступного року набув офіційного стандарт ECMA-262. З тих пір підтримка JS стала обов'язково для кожного браузера, і кожен великий виробник почав будувати свій рушій для підтримки JS. Протягом довгих років ці рушії розвивалися, змінювали один одного, перейменовувалися, і ставали основою для наступних.

До середини 2000-х JavaScript був стандартизований і дуже поширений, але його виконання було все ще повільним. Боротьба за швидкістю почалася з 2008, коли з'явився ряд нових рушіїв. На початку 2008 найшвидшим був Futhark від Opera. До літа Mozilla представила Tracemonkey, а Google запустив свій Chrome з новим JavaScript-компілятором V8. Незважаючи на велику кількість назв, усі вони намагалися робити одне і те ж, і кожен проект хотів вигідно

відрізнятися в швидкості виконання. Починаючи з 2008 компілятори поліпшувалися за рахунок оптимізацій свого дизайну, і між основними гравцями відбувалася гонка за побудову найшвидшого браузера.

Jesse Ruderman з Mozilla створив дуже корисний інструмент jsfunfuzz для тестування коректності компілятора. Брендан назвав це пародією на JavaScript-компілятор, так як його мета створювати найдивніші, але валідні конструкції, які відправляються на перевірку компілятору. Інструмент дозволив виявити численні крайні випадки і баги.

Популярні реалізацій JavaScript-engine:

- V8 – рушій з відкритим вихідним кодом, написаний на C ++, його розробкою займається Google.
- Rhino – цей рушій з відкритим кодом підтримує Mozilla Foundation, він повністю написаний на Java.
- SpiderMonkey – найперший з з'явився JS-engine, який в минулому застосовувався в браузері Netscape Navigator, а сьогодні - в Firefox.
- JavaScriptCore – ще один движок з відкритим кодом, відомий як Nitro і розробляється Apple для браузера Safari.
- KJS – JS-engine KDE, який розробив Гаррі Порті для браузера Konqueror, що входить в проект KDE.
- Chakra (JScript9) – engine для Internet Explorer.
- Chakra (JavaScript) – engine для Microsoft Edge.
- Nashorn – движок з відкритим кодом, який є частиною OpenJDK, яким займається Oracle.
- JerryScript – легкий engine для інтернету речей. Движок з відкритим кодом V8 був створений компанією Google, він написаний на C ++. Движок використовується в браузері Google Chrome. Крім того, що відрізняє V8 від інших engine, він застосовується в популярній серверної середовищі Node.js.

V8. Engine компілює JavaScript-код в машинні інструкції в ході виконання програми, реалізуючи механізм динамічної компіляції, як і багато

сучасних JavaScript-engines, наприклад, SpiderMonkey і v (Mozilla). Основна відмінність полягає в тому, що V8 не використовує при виконанні JS-програм байт-код або будь-який проміжний код.

SpiderMonkey – це інтерпретатор мови JavaScript для engine Gecko, написаний на мові C. Він використовується в різноманітних продуктах в рамках проекту Mozilla, включаючи браузер Firefox, і доступний на умовах потрійний ліцензії MPL / GPL / LGPL.

Rhino перетворює JavaScript скрипти в Java класи. Rhino працює і в компільованому і інтерпретованому режимах. Він призначений для використання в server-side додатках, тому в ньому немає вбудованої підтримки для об'єктів браузера, які зазвичай асоціюються з JavaScript.

Можна прослідкувати еволюцію компілятор та engine продуктивності.

У Firefox 3.5 (випущений 30 червня 2009 року) використовується техніка оптимізації, що передбачає «в деяких випадках поліпшення продуктивності в 20-40 разів» [3].

2 червня 2008 роки команда розробників WebKit представила SquirrelFish [4] – новий engine JavaScript, в якому досягалося значне поліпшення швидкості інтерпретування скриптів браузером Safari [5]. Цей engine був однією з нових можливостей Safari 4. Тестова версія з'явилася 11 червня 2008 року; в результаті engine був перейменований в Nitro.

З тих пір почалася гонка розробників браузерів по збільшенню швидкості engine JavaScript. З 2008 року пальму першості в ній утримує Google Chrome: це підтверджує безліч незалежних експериментів [6]. З появою Squirrelfish Extreme від розробників WebKit і Tracemonkey від Mozilla, продуктивність JavaScript в Google Chrome перестала розцінюватися як найвища [7]. Однак датський підрозділ Google розробив engine JavaScript V8, зі значно збільшеною продуктивністю JavaScript в Google Chrome 2.

Як правило, браузер має браузерні компілятори, що займаються рендерингом сторінок, і engine JavaScript, що спрощує тестування, перевикористання або використання в інших проектах. Наприклад, Caracan

використовується з Presto, Nitro з WebKit, Rhino і SpiderMonkey з Gecko, KJS з KHTML, за замовчуванням, ні з одним з браузерних engine не використовується. Іноді можливі інші комбінації, наприклад, V8 з WebKit в Google Chrome. Engine JavaScript дозволяє розробникам отримати доступ до функціональності (робота з мережею, з DOM, з зовнішніми подіями, з HTML5 video, canvas, storage), необхідної для управлінням веб-браузером.

Sunspider – інструмент тестування продуктивності браузера, який використовується для вимірювання продуктивності engine JavaScript в більш ніж дюжині тестів, кожен з яких заточений на окрему частину мови JavaScript. Sunspider не призначений для тестування можливостей, пов'язаних з чимось крім обчислень (HTML, CSS, робота з мережею).

Постановка завдання. Метою статі є аналіз та порівняння основних компіляторів мови JavaScript. Окрім того, досліджуються та описуються основні класи задач які можуть застосовуватися при використанні конкретного виду компілятора, структури та інформація про схему їх виконання.

Динамічна типізація Javascript – це те, що дозволяє одній і тій же властивості бути числом в одному місці і рядок – в іншому. На жаль, таке розмаїття призводить до численних додаткових перевірок типів в компіляторі, а код з умовними перевірками набагато довший і повільніший, ніж код, визначений для типів змінних.

Рішенням є метод виведення типів, який є у всіх сучасних JS-компіляторах. Компілятор робить припущення про типи даних властивостей. Якщо припущення вірне, він передає виконання типізовану JIT, яка генерує швидкий машинний код для цих ділянок. Якщо ж тип не збігається, то код передається нетипізованій JIT, для виконання на більш повільному коді з перевірками умов.

Інлайн кешування найпоширеніша оптимізація в сучасних JavaScript-компіляторах. Це не нова техніка (вперше була застосована 30 років тому для Smalltalk компілятора), але дуже корисна [1].

Інлайн кешування вимагає обидві техніки для своєї роботи: висновок типів і приховані класи. Коли компілятор виявляє новий клас, він кеширует його прихований клас разом з усіма визначеними типами. Якщо ця структура зустрічається пізніше, її можна швидко порівняти зі збереженим кешем. Якщо структура або тип даних змінилися, вони передаються в більш повільний узагальнений (generic) код або в деяких компіляторах виконується поліморфне інлайн кешування - генерація окремого кешу однієї структури для кожного типу даних.

Коли компілятор отримує структуру коду і типи даних в ній, стають можливими різноманітні додаткові оптимізації. Ось лише кілька прикладів:

- **inline expansion, або “inlining”**

Виклик функції є дорогою операцією, так як вимагає якого-небудь виду пошуку, а пошук може бути повільним. Ідея в тому, щоб помістити код тіла функції в те місце, де вона викликається. Це дозволяє уникнути зайвого розгалуження, і прискорює виконання, але за рахунок збільшення розміру виконуваного коду.

- **інваріантні зміни циклів, “підйом”**

Цикли є першими задачами при оптимізації. Прибравши непотрібні обчислення з циклу, можна сильно поліпшити продуктивність. Найпростіший приклад: цикл for за елементами масиву. Обчислювати довжину масиву на кожній ітерації невигідно, тому ця операція виноситься, "піднімається" за цикл.

- **згортка констант**

Обчислюються вирази зі сталими, а також висловлювання, що містять незмінні змінні.

- **видалення загальних підвиразів**

Аналогічно згортці констант, компілятор шукає вирази, що містять однакові обчислення. Ці вирази можуть замінюватися на змінні з розрахованими значеннями.

- **усунення мертвого коду**

Код, який не використовується, або його неможливо досягти. Немає сенсу оптимізувати тіло функції, яка жодного разу не використовується, її можна просто видалити.

- **ES.next**

Наступна версія специфікації EcmaScript (EcmaScript 6) вже давно в роботі. Однією з позначених цілей проекту є швидка компіляція. Обговорюється набір засобів, якими це можна досягти, включаючи типізацію, бінарні дані і типізовані масиви. Збірний код може безпосередньо вирушати в JIT, прискорюючи час компіляції і виконання.

Підтримка ES.next браузерами ще досить обмежена, але за цим можна стежити хоча б тут, також можна почати тестування за допомогою Traceur - компілятора ES.next в JavaScript, написаний на JavaScript.

- **WebGL**

JavaScript в браузері не обмежений маніпуляціями з DOM. Велика кількість сучасних браузерних ігор рендериться безпосередньо на canvas елементі сторінки, використовуючи стандартний 2D-контекст. Найшвидший спосіб рендерингу на canvas - WebGL, API забезпечує оптимізацію за рахунок перенесення дорогих операція на GPU, залишаючи CPU для логіки додатка.

WebGL в якомусь вигляді підтримується в більшості браузерів, в першу чергу в Chrome і Firefox. Користувачі Safari і Opera повинні спочатку включити відповідну опцію. Microsoft також оголосили про підтримку WebGL в IE11.

На жаль, навіть з повноцінно підтримкою браузерів, не можна гарантувати, що WebGL буде працювати однаково добре для всіх користувачів, так як це залежить ще і від сучасних драйверів GPU. Google Chrome є єдиним браузером, який пропонує альтернативне рішення, якщо цих драйверів не встановлено. WebGL - дуже потужна технологія. Крім питань безпеки, підтримка мобільних пристроїв дуже неоднорідна. І, звичайно, в старих браузерах немає ніякої підтримки [1].

Mozilla

- SpiderMonkey - найперший engine JavaScript, створений Бренданом Айхом в Netscape Communications.
- Rhino, що розробляється Mozilla Foundation engine JavaScript з відкритим вихідним кодом, повністю написаний на Java.
- Tamarin.

Google

- V8 - engine JavaScript з відкритим вихідним кодом, що розробляється датським відділенням компанії Google. Використовується в браузерах на основі Chromium, а також в Maxthon 3.

Інші

- KJS — ECMAScript / JavaScript-engine середовища робочого столу KDE, спочатку розроблений Гаррі портеньо для браузера Konqueror
- Narcissus — engine JavaScript з відкритим вихідним кодом, написаний Бренданом Айхом також на JavaScript
- Tamarin від Adobe Systems
- Nitro — engine JavaScript в Safari 4
- Chakra в Internet Explorer 9.
- Carakan від Opera Software, використовується в Opera, починаючи з версії 10.55

Висновки. В результаті проведеного аналізу було виявлено, що сама по собі мова Javascript (ECMAScript) – достатньо складна мова, її навряд чи вийде виконати читаючи рядки по порядку, тому вона інтерпретується за допомогою достатньо складного процесу.

Для інтерпретації була створена велика кількість engine технологій. Популярний engine V8 був спочатку створений для покращення продуктивності JavaScript в браузері. Саме з метою продуктивності V8 транслює JavaScript в більш ефективний машинний код замість інтерпретації. Він компілює код в байткод одразу, використовуючи JIT.

Engine JavaScript використовують декілька потоків виконання: основний потік робить те, що очікує – отримує код, компілює та виконує. Окрім того, існує потік, який займається тільки компіляцією, так, що основний потік може займатися своїми справами, в той час, коли інші потоки займаються оптимізацією.

Отже, основні компілятори мови JavaScript використовують full-codegen, який напряму транслює код, без будь-якої оптимізації. Це і дозволяє запускати код з високою швидкістю.

Список літератури

1. JavaScript Engine [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/JavaScript_engine
2. Internet Explorer 9 Beta Next – New IE9 Builds Every 8 Weeks [Електронний ресурс] / M.Oiaga – Режим доступу до ресурсу: http://news.softpedia.com/news/Internet-Explorer-9-Beta-Next-New-IE9-Builds-Every-8-Weeks-138013.shtml.
3. Firefox to get massive JavaScript performance boost [Електронний ресурс] / P.Ryan – Режим доступу до ресурсу: https://arstechnica.com/information-technology/2008/08/firefox-to-get-massive-javascript-performance-boost/
4. Announcing SquirrelFish [Електронний ресурс] / G.Garen – Режим доступу до ресурсу: https://webkit.org/blog/189/announcing-squirrelfish/
5. Apple Safari 4 [Електронний ресурс] / V.Lipskas – Режим доступу до ресурсу: https://webkit.org/blog/189/announcing-squirrelfish/
6. Speed test: Google Chrome beats Firefox, IE, Safari [Електронний ресурс] / S.Shankland – Режим доступу до ресурсу: https://www.cnet.com/news/speed-test-google-chrome-beats-firefox-ie-safari/
7. Firefox counters Google's browser speed test [Електронний ресурс] / S.Shankland – Режим доступу до ресурсу: https://www.cnet.com/news/firefox-counters-googles-browser-speed-test.