JavaScript ja virtuaalikoneet

Ville Lahdenvuo

Aine
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 5. lokakuuta 2015

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Ville Lahdenvuo

Työn nimi — Arbetets titel — Title

JavaScript ja virtuaalikoneet

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Aine	5. lokakuuta 2015	7

Tiivistelmä — Referat — Abstract

Tiivistelmä.

ACM Computing Classification System (CCS):
Software and its engineering → Virtual machines
Software and its engineering → Very high level languages

Avainsanat — Nyckelord — Keywords
JavaScript, virtuaalikone, suorituskyky

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

JavaScript on ohjelmointikieli, joka suunniteltiin ensisijaisesti verkkosivujen tekijöille. Sen tavoite oli täydentää Java-ohjelmointikieltä ja HTML-merkkauskieltä. JavaScript mahdollistaa monimutkaisten Internet-selaimissa suoritettavien sovellusten kirjoittamisen ilman selainlaajennuksia [8].

Selainten suosio sovellusalustana on kasvanut ja samalla JavaScriptin käyttö on lisääntynyt paljon. Kasvua siivittää se, että kaikissa kuluttajatietokoneissa on jokin selain ja sovellusten käyttämiseen riittää verkkosivuilla vierailleminen. Käyttäjän ei tarvitse asentaa mitään ennen sovelluksen käyttöä.

Selaimet ovat kehittyneet ja niiden käyttämiä teknologioita on standardisoitu. Näistä niin sanotuista Web-teknologioista, joihin JavaScript lasketaan, on tullut varteenotettava vaihtoehto moderniin sovelluskehitykseen. Web-teknologioiden käyttö ei kuitenkaan rajoitu vain selaimiin. Niillä on toteutettu esimerkiksi palvelinsovelluksia sekä kokonaan ilman selainta toimivia sovelluksia, kuten Atom-tekstieditori [4].

JavaScript on dynaaminen oliopohjainen kieli, mutta se tukee myös imperatiivista ja funktionaalista ohjelmointityyliä. JavaScript tarjoaa siis monia tapoja toteuttaa sama asia [2, 4.2.1.]. Muuttujat JavaScriptissä ovat dynaamisesti tyypitettyjä ja tämä helpottaa ohjelmakoodin kirjoittamista. Dynaamisuudesta seuraa kuitenkin myös ongelmia, sillä virtuaalikoneiden on vaikea ennustaa dynaamisia muutoksia ja tehdä järkeviä optimointeja, joilla suorituskykyä voitaisiin parantaa [1, s. 497].

Modernit JavaScript-virtuaalikoneet ovat kehittyneet paljon viime vuosina. Niihin on toteutettu monimutkaisia ja kekseliäitä menetelmiä suorituskyvyn parantamiseksi. Suuri osa virtuaalikoneiden optimoinneista perustuu oletukseen, että dynaamisuudesta huolimatta ohjelmat käyttäytyvät suorituksen aikana yleensä melko staattisesti. Keräämällä tyyppitietoa suorituksen aikana, virtuaalikoneet pystyvät esimerkiksi luomaan optimoitua konekoodia

osalle lähdekoodista. Optimoitavuus edellyttää, että ohjelmoija tietää miten asiat kannattaa toteuttaa. Jos hyödyntää dynaamisuutta liikaa, voi helposti tehdä koodia, jota virtuaalikone ei pysty optimoimaan.

JavaScriptiä kuitenkin kehitetään jatkuvasti ja siihen on tuotu muun muassa luokka- ja moduulijärjestelmät [2, 14.5. ja 15.2.]. Nämä auttavat yhtenäistämään erilaisia toteutustapoja ja mahdollistavat tällä tavoin aikaisempaa paremmin ennustettavan käytöksen. Kun käytetään luokkasyntaksia, tulee käytettyä yhtenäistä tapaa muodostaa objekteja. Ennustettavuudesta seuraa parempi optimoitavuus ja suorituskyky [1, s. 497].

JavaScriptin standardisoiminen, sen käytön lisääntyminen ja selainvalmistajien keskinäinen kilpailu suorituskyvystä on parantanut kielen asemaa ja mainetta. JavaScriptin rooli on muuttunut skriptikielestä yleiskäyttöiseksi ohjelmointikieleksi [2, 4.]. Kielen tulevaisuus näyttää lupaavalta. Siihen on tullut paljon ohjelmointia helpottavia ominaisuuksia ja tapoja välttää yleisiä sudenkuoppia, joihin varsinkin aloittelevat ohjelmoijat usein törmäävät.

2 Virtuaalikoneiden toiminta

Virtuaalikone on sovellus, tarjoaa todellisen tai hypoteettisen koneen toiminnallisuuksia ohjelmille, riippumatta laitteesta, jossa virtuaalikonetta suoritetaan. Niitä on kahdenlaisia, *järjestelmä-* ja *prosessivirtuaalikoneita* [9, s. 33]. Näiden erona on, että järjestelmävirtuaalikone tarjoaa kokonaisen käyttöjärjestelmän palvelut, kun taas prosessivirtuaalikone toteuttaa vaan yhden prosessin suorittamista varten tarvittavat rajapinnat.

Tässä tutkielmassa virtuaalikoneella tarkoitetaan ohjelmointikieliä suorittavia virtuaalikoneita, jotka ovat prosessivirtuaalikoneita. Niitä käytetään, koska abstrahoimalla fyysisen laitteen ominaisuuksia saavutetaan sen avulla toimiviin sovelluksiin helpommin alustariippumattomuus. Riittää tehdä virtuaalikoneesta alustariippumaton, jonka jälkeen kaikki sen avulla toimivat sovellukset ovat alustariippumattomia, sillä niille riittää, että virtuaalikone

toteuttaa samat toiminnallisuudet.

Virtuaalikonemalli parantaa myös tietoturvaa, sillä siinä suoritettava ohjelma pääsee käsiksi vain virtuaalikoneen tarjoamiin rajapintoihin. Tämä tarkoittaa, että pahaa tarkoittava ohjelma on eristetty todellisesta käyttöjärjestelmästä ja laitteistosta [9, s. 36].

2.1 Esimerkki: SpiderMonkey

Ensimmäisen JavaScript-virtuaalikoneen nimi on SpiderMonkey [3]. Se toteutettiin Netscape-selainta varten vuonna 1995 ja nykyään sitä ylläpitää Mozilla ja sitä käytetään muun muassa Mozillan Firefox-selaimessa. Nykyinen SpiderMonkey koostuu kolmesta pääkomponentista: *kääntäjä*, *tulkki* ja *roskienkerääjä* [7].

Kääntäjä huolehtii JavaScript koodin *jäsentämisestä* (parsing) ja kääntämisestä *tavukoodiksi*. Tavukoodi on eräänlainen *välikieli* (intermediate language), jota virtuaalikoneen on helpompi käsitellä kuin tekstimuotoista ohjelmakoodia.

Tulkin tehtävä on suorittaa tavukoodia. Tulkki siis lukee tavukoodia ja tekee tarvittavat toiminnot, jotka riippuvat alustasta. Tulkista on siis oltava oma versionsa jokaista tuettua alustaa varten. Todellisuudessa tulkkia käytetään vain suorituksen alkuvaiheessa keräämään tyyppitietoa. Usein kutsutut, niin sanotut "kuumat" funktiot, pyritään kääntämään optimoiduksi konekoodiksi *suorituksenaikaisella kääntäjällä* tai lyhyemmin *JIT-kääntäjällä* (Just-In-Time compiler) [lähde?].

Roskienkerääjän tehtävä on yksinkertaisesti poistaa muistista muuttujat ja oliot, joihin ohjelmassa ei enää viitata. Roskienkerääjän ansiosta ohjelmoijan ei tarvitse vapauttaa muistia itse, vaan järjestelmä hoitaa muistinhallinnan automaattisesti. Automaattinen muistinhallinta vähentää virheiden määrää, kuten muistivuotoja, mutta ei poista kaikkia ongelmia [lähde?].

2.2 Esimerkki: V8

Vuonna 2008 Google julkaisi uuden selaimen, Google Chromen, jonka oli tarkoitus parantaa verkkosovellusten käyttökokemusta [6]. Googlen kiinnostus käyttökokemuksen parantamisesta on ymmärrettävää, sillä yhtiöllä on paljon verkkopalveluita, jotka hyötyvät hyvästä suorituskyvystä. Näistä syistä Google päätyi toteuttamaan oman JavaScript-virtuaalikoneen V8:n.

Mielenkiintoisen V8:sta tekee se, että siinä ei ole lainkaan tulkkia. Sen sijaan V8 kääntää JavaScript koodin suoraan konekoodiksi ennen suorittamista. SpiderMonkeyn tapaan se kerää tyyppitietoa suorituksen aikana ja käyttää optimoivaa JIT-kääntäjää suorituskyvyn parantamiseksi [11].

2.3 Esimerkki: JavaScriptCore

JavaScriptCore on WebKit-nimisen Web-teknologioita toteuttavan ohjelmistokomponentin JavaScript-virtuaalikone. Apple käyttää WebKitiä ja JavaScriptCorea Safari-selaimessaan.

JavaScriptCore koostuu tulkista, yksinkertaisesta JIT-kääntäjästä sekä Googlen V8:n innoittamana optimoivasta JIT-kääntäjästä, jota he kutsuvat nimellä DFG-JIT. DFG tulee sanoista *Data Flow Graph*, joka kuvaa ohjelman suorituksenaikaisen tyyppitiedon tallentavaa tietorakennetta. Eli kuten aikaisempien esimerkkien kohdalla, virtuaalikone ensin kerää tyyppitietoa ja sitten generoi optimoitua konekoodia [10].

2.4 JavaScriptin tuomat haasteet

Kuten aikaisemmista esimerkeistä käy ilmi, virtuaalikoneet eivät voi suoraan generoida optimoitua konekoodia JavaScript-ohjelmista, sillä ei ole tietoa muuttujien tyypeistä. Prosessorin kannalta on hyvin tärkeää tietää tehdäänkö jokin operaatio kokonaisluvuille, liukuluvuille vai kenties merkkijonoille.

Automaattinen roskienkeruu on todella kätevä toiminnallisuus ohjelmoijan kannalta, mutta sen toteuttaminen hyvin on haastavaa. Ohjelman on

pysähdyttävä ja käytävä läpi muistin sisältöä ja vapauttaa muistialueita, jotka eivät ole enää käytössä. Selaimen tapauksessa tämä voi aiheuttaa verkkosovelluksen hidastumista ja huonontaa käyttökokemusta, varsinkin jos kyseessä on interaktiivinen toiminto tai animaatio.

3 Virtuaalikoneiden suorituskyky

On selvää, että JavaScript-koodin kääntäminen konekoodiksi on aina nopeampaa kuin tulkkaaminen, sillä tulkkaamiseen kuluu aina enemmän konekäskyjä kuin valmiin koodin suorittamiseen. Toki kääntäminen tuo ongelmaksi hitaan käännösvaiheen. Tämän takia modernit virtuaalikoneet sisältävät useamman kuin yhden kääntäjän eritasoisilla optimoinneilla. Tässä luvussa esitellään muutamia keinoja, joilla JavaScript-virtuaalikoneet ovat parantaneet suorituskykyä.

3.1 Piiloluokat

Piiloluokat (hidden classes) [5] ovat staattisia luokkia, joita virtuaalikone käyttää JavaScript-objektien kuvaamiseen. Jos objektia muutetaan, esimerkiksi lisäämällä uusi kenttä, luodaan sille uusi piiloluokka. Piiloluokkien hyödyt tulevat esiin, jos ohjelmassa on paljon samanlaisia objekteja, jotka voivat hyödyntää samaa piiloluokkaa. [Kuva tähän?]

Lähteet

1. Ahn, W., Choi, J., Shull, T., Garzarán, M.J. ja Torrellas, J.: *Improving JavaScript Performance by Deconstructing the Type System.* SIGPLAN Not., 49(6):496–507, kesäkuu 2014, ISSN 0362-1340. `http://doi.acm.org/10.1145/2666356.2594332`.

2. ECMA International: *ECMAScript® 2015 language specification*, kuudes painos, kesäkuu 2015. `http://www.ecma-international.org/ecma-262/6.0/`.

3. Eich, B.: *New JavaScript engine module owner.* `https://brendaneich.com/2011/06/new-javascript-engine-module-owner/`, vierailtu 3.10.2015 .

4. GitHub: *Atom — A hackable text editor for the 21st century.* `https://atom.io/`, vierailtu 16.9.2015 .

5. Google: *Chrome V8 design elements.* `https://developers.google.com/v8/design`, vierailtu 1.10.2015 .

6. Google: *Google Chrome: a new take on the browser.* Press Release. Google Inc, 2008. `http://googlepress.blogspot.fi/2008/09/google-chrome-new-take-on-browser_02.html`, vierailtu 4.10.2015 .

7. Mozilla: *SpiderMonkey Internals.* `https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Internals`, vierailtu 3.10.2015 .

8. Paolini, G: *Netscape and Sun announce JavaScript, the open cross-platform object scripting language for enterprise networks and the Internet.* Press Release. Sun Microsystems Inc, 1995.

9. Smith, J.E. ja Nair, R.: *The architecture of virtual machines.* Computer, 38(5):32–38, May 2005, ISSN 0018-9162.

10. Wingo, A: *JavaScriptCore, the WebKit JS implementation.* `https://wingolog.org/archives/2011/10/28/javascriptcore-the-webkit-js-implementation`, vierailtu 5.10.2015 .

11. Wingo, A: *V8: a tale of two compilers.* `https://wingolog.org/archives/2011/07/05/v8-a-tale-of-two-compilers`, vierailtu 4.10.2015 .