ZAVOD ZA PRIMIJENJENO RAČUNARSTVO
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
SVEUČILIŠTE U ZAGREBU

DART

j.i.n.x.

SEMINARSKI RAD IZ PREDMETA *PROGRAMSKE PARADIGME I JEZICI*

Zagreb, lipanj 2012

UVOD

Google Inc, američka tvrtka, javnosti je najpoznatija po svojoj internetskoj tražilici Google Search koja je vjerojatno najkorištenija tražilica na svijetu. No osim tražilice, ova tvrtka je otkako se pojavila na tržištu, lansirala i mnogo drugih stvari. Program Google Earth, servis GMail, internet preglednik Google Chrome, Google Docs te mobitel Galaxy Nexus (napravljen u suradnji sa Samsungom) samo su neki od uspješnica Googlea kojima se iz dana u dan privlači sve više korisnika. Najnovija inovacija Googlea je programski jezik Dart. Svjetlo dana je ugledao u desetom mjesecu prošle godine, a trenutno aktualna verzija je 0.09. U ovom seminarskom radu obradit će se razlozi stvaranja ovog programskog jezika, te pregled nekih njegovih prednosti u odnosu na druge programske jezike.

1. Razlozi stvaranja Darta

Programski jezik Dart namijenjen je za razvoj web aplikacija. Nastao je vođen željom da postane alternativa JavaScriptu, i napravljen je tako da pokriva nedostatke koje postoje u JavaScriptu. Dart je namijenjen kako malim, tako i velikim projektima.

Na prvom predstavljanju razvijenog jezika, član razvojnog tima Lars Bak pojasnio je glavne ciljeve projekta Dart. Pri stvaranju jezika, vodilo se za idejom stvaranja strukturnog, ali i fleksibilnog jezika koji služi za web programiranje. Jezik je napravljen tako da programerima bude poznat i prirodan i tako omogućio brzo usvajanje. Naravno, osim stvaranja relativno jednostavnog jezika koji se lako usvaja, bilo je potrebno paziti i na tehničke značajke. Tako je jedan od glavnih ciljeva razvojnog tima bio napraviti programski jezik koji će pružati visoke performanse na svim modernim internet preglednicima i okruženjima; od malih mobilnih uređaja, sve do izvršavanja na velikim internet poslužiteljima.

2. Prednosti programskog jezika Dart

2.1. Opcionalno korištenje tipova

Programski jezik Dart omogućuje programiranje s korištenjem tipova, ali i pisanje programa bez dodjeljivanja strogog tipa varijabli. U programskom jeziku JavaScript varijablama se ne dodjeljuju tipovi i razvojni programeri JavaScripta tvrde da tipovi „uništavaju“ fleksibilnost programiranja. S jedne strane imaju pravo, ali programiranje većih projekata s tolikom fleksibilnošću može ozbiljno naštetiti programu. Dart dopušta oba načina programiranja. Ukoliko se želi programirati bez tipova, programira se kao i u JavaScriptu, a želi li programer koristiti stroge tipove, potrebno je samo omogućiti provjeru tipova (eng. *type checker*) i početi programirati. U nastavku je dan primjer kako se program ponaša kada se ne koristi provjera tipova, a kako kada se tipovi provjeravaju korištenjem *type checkera*.

Programski kod:

```
greeting(String name) {
    print("hello ${name}");
}

main() {
    int n = 7;
    greeting(n);
}
```

Rezultat izvršavanja programa bez provjere tipova:

```
hello 7
```

Rezultat izvršavanja programa s uključenom provjerom tipova:

```
Failed type check: type Number is not assignable to type String
```

2.2. Korištenje samo jedne mogućnosti za evaluaciju izraza kao false

Još jedno poboljšanje u odnosu na JavaScript odnosi se evaluaciju izraza unutar *if* uvjeta. JavaScript će za nekoliko različitih vrijednosti varijabli donijeti odluku da se radi o *false* vrijednosti. To su: *false*, *null*, *undefined*, prazan string, broj nula i broj NaN (*Not a Number*). Sve ostalo će dati vrijednost *true*, pa čak i prazan niz ili lista (što je malo kontradiktorno s pravilom JavaScripta da prazan string u *if* uvjetu vraća false). U Dartu samo će se samo jedna vrijednost evaluirati kao *false* – boolean vrijednost *false*. Tako se u *if* dio koda neće ući ukoliko je u *if*-izrazu neki od sljedećih izraza:

```
if(false)
```

```
a = "notEmptyString";
if(a=="")
```

```
if(0==1)
```

Općenito, kada Dart očekuje boolean vrijednost (a u *if* uvjetu je uvijek tako), ako vrijednosti nije *true*, onda je *false*. To je napredak u odnosu na JavaScript jer je puno bolje pretpostaviti da je nešto *false* nego obratno zbog mogućih previda pri programiranju.

2.3. Poboljšanje u radu s DOM-om

Jedna od velikih prednosti Darta je lakši rad s DOM-om (eng. *Document Object Model*). Prva od nekoliko napravljenih promjena je bilo uklanjanje prefiksa HTML iz većine imena tipova; tako je na primjer `HTMLElement` postao `Element`. Također, preoblikovana su imena koja se koriste za odnos čvorova - `childNodes` je postao `nodes`, a tip `children` je postao `elements`. Dosta imena je optimizirano kako bi se ona imena koja se najčešće koriste bila što sažetija.

Današnji DOM ima jako puno metoda koje služe za dohvaćanje raznih stvari. Neke od takvih metoda su:

- getElementsById() ;
- getElementsByTagName() ;
- getElementsByName() ;
- getElementsByClassName();
- querySelector() ;
- querySelectorAll() ;
- document.links ;
- document.images ;
- document.forms ;
- document.scripts ;
- formElement.elements :
- selectElement.options ;
- ...

Razvojni inženjeri Darta su sve takve metode, po uzoru na jQuery, sveli na samo dvije metode:

- query() i
- queryAll() .

Usporedba metoda koje se koriste JavaScript programeri, i njihovih ekvivalenata u Dart programskom jeziku dana je u sljedećoj tablici:

JavaScript	Dart
elem.getElementById('foo');	elem.query('#foo');
elem.getElementsByTagName('div');	elem.queryAll('div');
elem.getElementsByName('foo');	elem.queryAll('[name="foo"]');
elem.getElementsByClassName('foo');	elem.queryAll('.foo');
elem.querySelector('.foo .bar');	elem.query('.foo .bar');
elem.querySelectorAll('.foo .bar');	elem.queryAll('.foo .bar');

Tablica 1: Usporedba metoda za rad s DOM-om

Još jedna promjena koju je Dart uveo jest promjena načina na koji su event handleri (mehanizmi koji reagiraju na neki događaj na unaprijed određeni način) povezani. DOM ima dva načina na koja može raditi s događajima (eventima). Stariji način je da se samo jedan handler može povezati i to tako da se postavi jedno od `on__` svojstava izravno na element. Moderniji način izvedbe je da se iskoriste metode `addEventListener()` i `removeEventListener()`. Tim pristupom se omogućuje da više slušača (eng. *listeners*) čeka na isti event. Također, tip događaja se identificira preko njegovog imena (stringa) što može biti podložno pogreškama.

U Dartu su se ukinula `on__` svojstva za element, i stvorena je nova klasa ElementEvents. Za svaki poznati tip događaja, u toj klasi postoji svojstvo - npr. `click`, `mouse down`, i slična. Svako od tih svojstava je objekt događaja koji može dodavati ili uklanjati slušače i upravljati samim događajima. Spajanje na DOM je napravljeno tako što je elementima dano svojstvo `on` kojim se obavlja pristup. Tako su sve funkcionalnosti vezane uz događaje sadržani u jednom svojstvu. Usporedba provođenja *event handlinga* u DOM-u i onoga koji se provodi u Dartu prikazan je sljedećim kodovima:

```
// DOM
elem.addEventListener(
      'click',(event) =>
print('click!'), false);

elem.removeEventListener(
    'click', listener);
```

```
// Dart
elem.on.click.add(
     (event) =>
print('click!'));

elem.on.click.remove(listener);
```

2.4. Metoda `noSuchMethod`(..)

Ova metoda je izvrsno svojstvo koje programski jezik Dart omogućava. Naime, ako se nad objektom pozove funkcija koja ne postoji, neće doći do bacanja iznimke, već će se poziv preusmjeriti na posebnu metodu `noSuchMethod(..)` koju svi objekti nasljeđuju iz klase `Object`.

Kada se na takav način pozove naslijeđena metoda `noSuchMethod`, prema originalnoj implementaciji, metoda će vratiti `NoSuchMethodException`, što je u stvari isto što bi se dogodilo i da metoda `noSuchMethod` ne postoji. Ali, ono što je prava svrha metode, je to da ju se može nadjačati (eng. *override*) kako bi izvršavala nešto umjesto/uz bacanje iznimke.

Primjer mogućeg nadjačavanja metode dan je u nastavku.

```
  noSuchMethod (InvocationMirror msg){
        // throw NoSuchMethodException(...);
        // is line in not-overrided method
        if (msg.memberName=='foo') {
5           msg.invokeOn(bar());
        }
        else {
8           bar.call.apply(msg.arguments);
        }
  }
```

U navedenom primjeru se pri pozivanju metode koja nije implementirana, poziva `noSuchMethod` koja će u petoj liniji koda proslijediti poziv nekom drugom objektu (rezultatu `bar()` metode) ili će pak, izvršiti liniju 8 koda i proslijediti samo parametre nekoj drugoj funkciji.

2.5. Višedretvenost

Jedna od vrlo korisnih Dartovih mogućnosti koju JavaScript nema, jest korištenje višedretvenosti. U Dartu se ne koristi pojam dretvi, već su razvojni inženjeri stvorili pojam „*Isolates*“, koji u hrvatskom jeziku još nema adekvatan prijevod, pa ću u daljnjem tekstu koristiti navedeni engleski izraz.

Dakle, u Dart je ugrađena klasa *Isolates* čija se funkcionalnost može razdijeliti na dvije funkcionalnosti: klasa služi kao predložak za stvaranje nove *Isolate* te kao ulazna točka za novostvorenu *Isolate*. *Isolates* se dijele na lake i na teške. Oba tipa su asinkrona i međusobne komunikacije idu preko portova. Razlika je u tome što se teške *Isolates* nalaze u svojoj vlastitoj dretvi, dok se lake nalaze u istoj dretvi u kojoj se nalazi *Isolate* koja ih je stvorila. Svaka *Isolate* ima svoj vlastiti stog – sve vrijednosti pohranjene u memoriju, uključujući i vrijednosti globalnih varijabli, dostupne su samo jednoj *Isolate* u sklopu koje su pohranjivane u memoriju. Jedini način komunikacije između *Isolatesa* je prenošenjem poruka preko portova.

Da bi se *Isolates* mogle koristiti potrebno je uključiti knjižnicu `dart:isolate` na sljedeći način: `#import('dart:isolate');`. U primjeru u nastavku opisano je kako se može stvoriti i koristiti *Isolate*.

Nakon uključivanja knjižnice koja nam omogućuje rad, potrebno je i napisati funkciju koja će se izvoditi u novostvorenoj *Isolate*. U ovom primjeru to je funkcija `process()`.

```
process() {
  port.receive((var message, SendPort replyTo) {
      print ("Got message: ${message}");
      replyTo.send("close");
      port.close();
  });
}
```

Kao što je ranije spomenuto, *Isolates* komuniciraju preko portova. Kada se stvori nova *Isolate*, ona će preko porta primiti poruku te port na koji treba odgovoriti. U tijelu

metode `receive` sadržan je kod koji se treba izvršiti u stvorenoj *Isolate* kao i povratna poruka na predani port. Nakon izvršavanja funkcije i odgovora da je *Isolate* završila s radom, port je potrebno zatvoriti. Ako se on ne zatvori, postoji opasnost da će se kod vrtjeti zauvijek i napraviti probleme u memoriji računala.

Sada kada postoji funkcija koja će se izvršavati u stvorenoj *Isolate*, potrebno je objasniti i kako se zapravo ona stvara.

```
main() {
   port.receive((var message, SendPort replyTo) {
     print("Got message: ${message}");
     port.close();
   });

   SendPort sender = spawnFunction(process);
   sender.send("start", port.toSendPort());
}
```

U ovom primjeru nova *Isolate* se stvara u glavnom programu koji također predstavlja jednu *Isolate* koja ima svoj port spreman za korištenje. Na ovaj način se može postići ispisivanje poruke u glavnom programu, ali nakon zatvaranja njegovog porta, više ne možemo ništa ispisivati unutar *Isolate* glavnog programa. No moguće je stvoriti novu *Isolate* korištenjem metode `spawnFunction` koja kao parametar prima ranije definiranu funkciju `process`. Metoda `spawnFunction` vraća port na koji se može poslati poruka. Poruka može biti tipa null, bool, num, double, String, kao i tip List i Maps koje sadrže navedene primitivne tipove. Tip object je dopušten, ali kod korištenja lake *Isolate*. Kada se poruka šalje preko metode `send`, potrebno je predati i drugi argument koji označava port na koji se prima odgovor od stvorene *Isolate*.

ZAKLJUČAK

Dart je jezik koji ima značajne prednosti i poboljšanja u odnosu na jezik JavaScript koji je danas najrašireniji za razvoj web aplikacija. No to je i za očekivati od jezika koji je stvoren šest godina kasnije. Štoviše, možda su se mogle doraditi i još neke sitnice. Ali Dart je još uvijek programski jezik u razvoju, njegove knjižnice redovito prolaze kroz nadopunjavanje i ispravke. Razvojni inženjeri Darta se oslanjaju i na programere koji su Dart prihvatili i počeli se njime služiti te imaju konstruktivne kritike na račun jezika. Glavni problem prihvaćanja Darta kao programskog jezika koji u potpunosti može zamijeniti JavaScript, jest to što je JavaScript kao prvi prihvaćen jezik tog tipa raširen posvuda – od mobilnih uređaja i tableta do modernih internet preglednika. Dart se oslanja na preglednik Google Chrome koji bi progurao Dart, ali Dart i dalje funkcionira tako da se ponekad treba prevoditi u JavaScript, što znači, da ga barem u dogledno vrijeme, neće moći u potpunosti zamijeniti.

LITERATURA

[1] DART, http://www.dartlang.org/

[2] Gentle introduction to Dart with Gilad Bracha, http://blog.japila.pl/2011/12/gentle-introduction-to-dart-with-gilad-bracha/

[3] Type Checking, http://www.dartforce.com/doc/typing101

[4] Improving the DOM, http://www.dartlang.org/articles/improving-the-dom/

[5] Emulating Functions in Dart, http://www.dartlang.org/articles/emulating-functions/

[6] Darts new Isolate API, http://www.grobmeier.de/darts-new-isolate-api-dartlang-17042012.html

[7] Dart (programming language), http://en.wikipedia.org/wiki/Dart_(programming_language)

[8] Truthy and falsy in JavaScript, http://11heavens.com/falsy-and-truthy-in-javascript