ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

IL LINGUAGGIO DART PER LA PROGRAMMAZIONE DI APPLICAZIONI WEB STRUTTURATE

Elaborata nel corso di: Sistemi Distribuiti L-A

Tesi di Laurea di:
NATALINA ARLOTTI

Relatore:
Prof. ALESSANDRO RICCI

ANNO ACCADEMICO 2010 –2011
SESSIONE III

PAROLE CHIAVE

Web 2.0

Html 5

Dart

Indice

Introduzione

In un contesto dinamico e spesso confuso come quello del Web, che deve la sua forza alla facilità con cui è possibile usufruirne, sia per consultare che per produrre il materiale che vi è contenuto, stare al passo con i cambiamenti che l'evoluzione tecnologica permette e che l'evoluzione della società richiede, è compito assai arduo.

In questi ultimi anni, abbiamo assistito ad una rapida crescita nella richiesta di strutture e applicazioni che consentissero all'utente una maggiore interattività rispetto al Web del passato, portando alla proliferazione di blog, forum, social network e così via. Le pagine Web, inoltre, sono sempre più migrate, da un semplice elenco di informazioni, verso una collezione di funzionalità finalizzate all'interazione con l'utente, per riuscire a soddisfare meglio le esigenze di quest'ultimo.

A supporto di questa nuova concezione del Web, sono nate, separatamente, innumerevoli tecnologie dal carattere più disparato: linguaggi di programmazione, linguaggi di scripting, fogli di stile, etc. Ognuna di queste tecnologie, però, si è sviluppata concentrandosi nell'ambito applicativo per il quale era stata pensata, ignorando più o meno integralmente le altre tecnologie con le quali sarebbe poi andata a coesistere.

Questo ha portato ad una situazione in cui le potenzialità di interazione con i contenuti da parte dell'utente sono enormi, ma che risulta essere incredibilmente confusa e mal strutturata, soprattutto da un punto di vista ingegneristico.

Negli ultimi tempi sono sempre più i server web finalizzati a fornire servizi su richiesta, (editor di testo, servizi di posta o messaggistica istantanea, archiviazione di documenti, foto o video, etc.) eliminando la necessità da parte dell'utente di dotare la propria macchina di programmi particolari in grado di svolgere tali servizi, spostando il carico computazionale lato server. Tutto ciò è fatto integrando diverse funzionalità fornite da svariate tecnologie, che spesso non sono affatto adatte a condividere il proprio spazio operativo le une con le altre. In particolare, il compito di organizzarne l'interazione è decisamente arduo, per il fatto che agli utenti piace poter scegliere di quali set di ser-

vizi usufruire, impedendo quindi una predizione su quali componenti andranno a interagire.

In questa tesi verrà analizzata l'evoluzione che ha portato al Web 2.0, mettendo in evidenza le necessità che hanno condotto a tale evoluzione, le strutture che da essa sono scaturite e in che modo le tecnologie si sono adattate nel corso del tempo. Sarà presentato, quindi, il linguaggio di nuova concezione per la programmazione web strutturata, Dart. Questo linguaggio si prefigge di affiancare e, in un secondo tempo, sostituire il linguaggio Javascript, che seppur usato in maniera intensiva ed estensiva, manca di una valida struttura sintattica, rendendolo inadatto a molti degli utilizzi per cui è impiegato.

Il linguaggio, inoltre, sarà in grado di fornire un supporto per l'integrazione di svariate funzionalità che allo stato attuale sono gestite più o meno singolarmente da altrettante tecnologie, rendendo possibile un elevato grado di compatibilità tra tutti i possibili servizi web e applicazioni lato client sviluppate utilizzando il linguaggio.

Capitolo 1

Evoluzione del Web

1.1 Le origini

Internet si sviluppò come logica conseguenza della nascita delle prime reti di telecomunicazioni. Fu pensato, infatti, come una rete informatica in grado di permettere la comunicazione tra vari utenti sparsi nel mondo. Nacque nel 1962, come un progetto finanziato dal governo degli Stati Uniti d'America, quando l'aeronautica militare chiese all'azienda californiana RAND Corporation come avrebbe potuto mantenere il controllo delle armi e dei bombardieri in caso di attacco nucleare.

L'ingegnere informatico Paul Baran, elaborò due sistemi per creare una rete di comunicazioni resistente a un eventuale attacco:

- costruire collegamenti alternativi tra i vari campi (rete decentrata), in modo che se alcuni di essi fossero stati interrotti, sarebbe sempre stata possibile la comunicazione tra gli altri nuclei militari;

- dividere il messaggio in varie parti (pacchetti) ognuno dei quali avrebbe seguito la propria strada verso la destinazione.

In questo modo l'interruzione di un collegamento non avrebbe comportato la perdita dell'intero messaggio ma solo di qualche pacchetto, mentre il resto del messaggio sarebbe arrivato a destinazione. Questa divisione del messaggio, in pacchetti indipendenti, esiste tuttora ed è alla base dei protocolli per l'invio di messaggi in rete, attualmente viene chiamata 'packet switching'.

Nel 1968, l'agenzia ARPA (Advanced Rearch Project Agency), assegnò un contratto alla società BBN per realizzare una rete privata sulla base del documento redatto da Baran. Il progetto prese il nome

di ARPANET. Appena un anno dopo nacque il primo nucleo di Arpanet che collegava tra loro quattro università americane: UCLA, SRI a Stanford, Università dello Utah, Università della California a Santa Barbara. Tutte le località erano connesse da linee dedicate a 50 Kbps.

Negli anni successivi si aggiunsero altri 37 nodi anche grazie al contributo di Ray Tomlinson che, per scambiare opinioni con i suoi colleghi, introdusse il primo programma di posta elettronica che permetteva agli utenti di ARPANET di scambiare messaggi utilizzando il simbolo '@' con il significato di 'presso'.

Il primo gennaio 1983 tutti i computer connessi alla vecchia rete ARPANET passarono insieme ad utilizzare il protocollo TCP/IP che diventò il protocollo di base della rete dell'ARPA. Durante lo stesso anno fu sviluppato il protocollo DNS (Domain Name System) che permetteva l'uso di nomi al posto degli indirizzi numerici per l'identificazione degli host, in questo modo gli indirizzi poterono diventare nomi più facili da memorizzare per l'utente.

Nel 1984 ARPANET fu suddivisa in due reti distinte. La prima, usata esclusivamente a scopi militari, fu chiamata MILNET, la seconda, invece, conservò il nome della rete originale. Con il passare del tempo l'esercito si disinteressò sempre più al progetto fino ad abbandonarlo definitivamente nel 1990. Tale progetto rimase quindi sotto il pieno controllo delle università, diventando un utile strumento per scambiare le conoscenze scientifiche e per comunicare.

Grazie ai primi tentativi di sfruttamento commerciale e a una serie di servizi offerti, nei primi anni novanta ebbe inizio il vero boom di ARPANET, nel frattempo rinominata INTERNET.

Negli stessi anni nacque una nuova architettura in grado di semplificare enormemente la navigazione: il World Wide Web.

1.2 Da Internet al Web

L'impulso finale allo sviluppo della rete fu dato dalle esigenze di confronto e scambio di dati e informazioni all'interno della comunità scientifica.

Nei primi anni 80, cominciarono ad arrivare presso il CERN (Conseil Europeen pour la Recherche Nucleare) a Ginevra in Svizzera, numerosi scienziati da tutto il mondo, impegnati nella ricerca sulla materia. Lo scambio d'informazioni divenne un problema di vitale importanza: tra le persone impegnate nella ricerca, infatti, quasi nessuno possedeva un terminale. I programmi erano impostati nella sala centrale di controllo e questo era l'unico punto di accesso alle informazioni, per cui gran parte del lavoro degli scienziati rimaneva inac-

cessibile ai colleghi ed era un problema anche collegare i vari progetti ai relativi autori.

Proprio per risolvere questi problemi, Tim Berners-Lee , che a quel tempo lavorava come consulente al CERN nel campo dell'ingegneria del software, realizzò un programma di appunti chiamato Enquire che serviva per immagazzinare informazioni usando associazioni casuali.

Enquire consentiva, attraverso un archivio (database) in cui i file erano archiviati per titolo e tipo, di spostarsi all'interno dello stesso file o da un file all'altro servendosi di un navigatore (browser) battezzato World Wide Web, limitato però alla sola visualizzazione testuale.

Il Web non fu l'unico browser sviluppato in quegli anni, alcuni studenti dell'Università del Minnesota avevano ideato un protocollo di trasmissione, chiamato Gopher, che consentiva di effettuare ricerche nel Gopher Space. L'utilizzo di Ghoper però non decollò mai a causa del tentativo di far pagare agli utenti l'utilizzo di questa nuova tecnologia.

Nel caso del WEB, invece, il CERN annunciò di voler rendere di pubblico dominio il protocollo ed il codice del WEB, per fare in modo di favorire lo sviluppo della comunicazione, decretandone così il successo.

1.3 Oltre il Web 1.0

Il Web degli anni '90, denominato Web 1.0, presentava una versione statica dei siti Internet molto diversa da quella odierna.

L'utente poteva navigare tra i vari siti senza però poter interagire con essi. Il web era concepito solamente come un modo per visualizzare documenti ipertestuali in formato HTML che rendeva testo e contenuto inseparabili. L'utente era quindi passivo, gli era soltanto permesso l'invio di posta elettronica e solo in formato testuale. Un sito poteva contenere informazioni utili ma non c'era ragione per un visitatore di ritornavi in un secondo momento.

I visitatori potevano solamente consultare questi siti: non potevano contribuire in alcun modo. Secondo la filosofia del Web 1.0 inoltre, le applicazioni erano esclusivamente proprietarie. Le compagnie sviluppavano software che gli utenti potevano scaricare, ma non permettevano loro di vedere in che modo funzionavano o di cambiare tali applicazioni.

In pochi anni l'evoluzione del Web è stata notevole: si è passati in un primo momento, con la versione 1.5, all'integrazione dei database, ai primi forum e blog molto semplici, e successivamente, con la versione 2.0, all'evoluzione delle community, ai social network e all'introduzione

dei wiki, che consentono all'utente di interagire nello sviluppo dei siti web.

'Il Web 2.0 è la rete come piattaforma. Attraverso tutti i dispositivi collegati, le applicazioni Web 2.0 permettono di ottenere la maggior parte dei vantaggi intrinseci della piattaforma, fornendo il software come un servizio in continuo aggiornamento, che migliora più le persone lo utilizzano, sfruttando diverse combinazioni di dati provenienti da sorgenti multiple.

Gli utenti forniscono i propri contenuti e servizi in un modo che permette il loro riutilizzo da parte di altri utenti, creando una serie di effetti attraverso un'architettura della partecipazione e andando oltre la metafora delle pagine del Web 1.0 per produrre esperienze più significative.' - Tim òReilly

Il progressivo passaggio dal web 1.0 al 2.0 ha avuto conseguenze positive anche dal punto di vista sociale. La semplicità di utilizzo, il poter pubblicare anche senza conoscere linguaggi complicati e la possibilità di separare le informazioni dalla forma, ne hanno decretato il successo.

Con Web 1.0 solo gli utenti esperti erano in grado di creare un sito in quanto era necessaria una buona conoscenza sia di HTML sia di linguaggi informatici e questo ne limitava la diffusione. I blog, invece, hanno semplificato la creazione di pagine interattive: chiunque può pubblicare sul proprio sito contenuti dinamici di grande effetto senza necessariamente possedere particolari conoscenze tecniche, questo ha dato la possibilità a tutti di condividere informazioni con altri utenti.

Il blog è un sito dove sono pubblicati dall'autore, in ordine cronologico, articoli condivisi con i suoi contatti. Principalmente viene usato dall'utente come diario personale, ma non mancano pagine di giornalisti, scrittori e personaggi pubblici. La struttura del sito è basata su un programma di pubblicazione guidato che permette di creare con facilità pagine web che possono essere modificate e personalizzate dal punto di vista grafico. I visitatori del blog creano una comunità interattiva che nasce dai commenti ai post, dalle riflessioni e dalle opinioni personali scambiate.

Con le tecnologie wiki gli utenti possono non solo recuperare informazioni, ma modificarle e aggiungerne altre. E' un sistema particolarmente collaborativo dove diverse persone aggiornano i contenuti esistenti come meglio credono. Questo processo fa si che programmi come Wikipedia siano in perenne aggiornamento e sviluppo. Wikipedia è un enciclopedia online usufruibile da tutti che tratta svariati argomenti in ben 250 lingue differenti. Essendo tutti potenziali scrittori di questa enciclopedia, l'attendibilità delle sue informazioni non è sempre accertata. Sta agli utenti stessi garantirne la veridicità.

Un altro importante sito creato dagli utenti è YouTube dove chiunque può visualizzare e votare filmati di altri utenti ed inserirne di nuovi. Questo è un fenomeno internazionale ormai estremamente diffuso.

Esistono inoltre programmi di e-commerce come Ebay, dove i navigatori possono acquistare o vendere oggetti di ogni tipo o partecipare ad aste.

1.4 Il Web 2.0

Il Web 2.0 è l'insieme di tutte quelle applicazioni online che consentono un elevato livello d'interazione tra sito e utente e che permettono partecipazione e collaborazione nello scambio d'informazioni all'interno del Web (blog, forum, chat, Wikipedia, Youtube, Facebook, Myspace, Twitter, Gmail, etc.).

I siti del Web 2.0 permettono agli utenti un grado d'interattività enormemente maggiore rispetto a quello che accadeva in passato, trasformando il ruolo degli utenti, da semplici visitatori del sito a collaboratori attivi. Un valido esempio di questa interattività sono i social network, i blog e i progetti wiki.

Il termine 2.0 ci suggerisce che siamo in presenza di una nuova versione del World Wide Web; in realtà si tratta di una serie di cambiamenti su come gli sviluppatori e gli utenti utilizzano il Web e non di un aggiornamento di specifiche tecniche.

I siti Web 2.0 consentono molto di più che recuperare informazioni, forniscono infatti all'utente interfacce e software accessibili tramite il browser. I siti stimolano gli utenti a utilizzare le informazioni in esso contenute ma, soprattutto, a incrementare il loro valore fornendo il proprio contributo e permettono, allo stesso tempo, di esercitare un certo grado di controllo su di esso. Il Web 2.0 offre, indipendentemente a tutti gli utenti, la stessa libertà di contribuire; questo sicuramente incoraggia la collaborazione ma permette anche comportamenti illeciti da parte di alcuni utenti.

La tecnologia principalmente utilizzata nei browser è Asynchronous Javascript and XML (Ajax). L'utilizzo di Javascript permette di caricare e scaricare nuove informazioni dal server senza bisogno di ricaricare completamente la pagina. Le informazioni recuperate da una richiesta Ajax sono tipicamente formattate in XML o JSON (Javascript Object Notation) e, visto che questi due formati sono compresi nativamente da Javascript, un programmatore li può facilmente utilizzare per trasmettere informazioni da e verso la propria applicazione web; Javascript, ricevute le informazioni via Ajax, utilizza il

DOM (Document Object Model) per aggiornare la pagina, rendendo le pagine molto di più simili a delle applicazioni desktop.

Un'altra tecnologia largamente utilizzata è Adobe Flex che, rispetto a Javascript, permette di gestire più facilmente griglie di dati e in generale operazioni molto pesanti. Le applicazioni programmate in Flex sono compilate e visualizzate in Flash all'interno del browser. Utilizzando Flash è possibile riprodurre file audio e video rendendo così possibile la creazione di siti in cui i dati multimediali sono integrati nello standard HTML, cosa non possibile in precedenza.

Lato server, il Web 2.0 usa molte tecnologie già esistenti come PHP, Ruby, Perl, Python, JSP, permettendo agli sviluppatori di manipolare dinamicamente informazioni provenienti da file e database. Il vero cambiamento è nella maniera in cui tali informazioni sono formattate.

Alle origini del web non c'era la necessità di far comunicare tra loro siti differenti. Con l'arrivo del Web 2.0, condividere dati tra siti è diventata una necessità fondamentale, per cui si rende necessario che tali dati vengano rappresentati attraverso degli standard comprensibili a una macchina come XML o JSON, rendendo possibile a un sito integrare una propria parte utilizzando informazioni provenienti da un altro sito, portando a informazioni più facilmente reperibili e categorizzate.

I siti Web 2.0 sono caratterizzati inoltre dalle seguenti caratteristiche e tecniche che prendono il nome di 'slates':

- Search - recuperare informazioni tramite una parola chiave
- Links - collegare le informazioni
- Authoring - creare e aggiornare i contenuti porta alla partecipazione di innumerevoli autori
- Tags - parola chiave per facilitare la ricerca
- Extentions - software che rende il Web una piattaforma applicativa invece che un semplice server di documenti
- Signals - utilizzo di tecnologie come RSS per notificare all'utente cambiamenti nel contenuto

L'informatica ha manifestato una evoluzione estremamente rapida: in pochi anni, si è passati da una rete sprovvista di interfacce grafiche dove solo un utente esperto riusciva a trovare le informazioni desiderate tra pagine piene di codici, a un sistema di comunicazione e informazione alla portata di tutti.

Già si parla di Web 3.0 per denotare la prossima evoluzione del web dove i dati verranno raccolti in un database, chiamato DataWeb, per poi essere riutilizzati e pubblicati più volte dagli utenti. Il Web 3.0 è sempre più vicino all'intelligenza artificiale, capace di interagire con il web e di capire le informazioni in esso contenute. Le applicazioni di questo sistema saranno leggere per cui non non si renderà necessario l'utilizzo di computer potenti e hard disk di grandi dimensioni. Sfrutteranno i social network senza appoggiarsi a un server centrale.

La nuova versione di HTML, la quinta, ha il compito di assimilare le nuove tendenze del Web, accogliendo i numerosi e importanti cambiamenti che nell'ultimo decennio hanno portato alla nascita del Web 2.0. Negli anni più recenti il web come abbiamo visto si è evoluto, raggiungendo uno stato più dinamico di quanto non avesse in precedenza. Lo sforzo di HTML 5 è di mantenere e far evolvere HTML per soddisfare i bisogni degli autori web del ventunesimo secolo.

1.5 Dal Web 2.0 al cloud computing

Con il termine cloud computing si intende un insieme di tecnologie che, sotto forma di servizi offerti da un provider a un utente, consentono di memorizzare, archiviare ed elaborare dati, grazie all'utilizzo di risorse hardware e software (CPU, reti, server, storage, applicazioni e servizi) distribuite.

Un'applicazione desktop è fisicamente eseguita sul computer connesso in rete e utilizza le risorse della macchina su cui è installata e per le applicazioni che hanno un considerevole carico computazionale, sono richiesti requisiti hardware ben definiti. Al contrario, poichè un'applicazione cloud è resa fruibile attraverso la connessione a un altro sistema tramite un browser, il carico computazionale è quasi interamente gestito dal service provider che offre il servizio, riducendo i requisiti hardware della macchina che lo utilizza.

Non sarà più necessario quindi un hardware particolarmente potente perchè migreranno online tutte quelle attività che fino a poco tempo fa erano necessariamente svolte in locale. I dati che erano salvati sui computer saranno decentrati su vari server e accessibili grazie al browser (la distribuzione su server diversi è opportuna per motivi di sicurezza, bilanciamento di carico, velocità e collocazione geografica).

I vantaggi del cloud computing sono diversi:

- diminuzione dei costi per il computer, non c'è bisogno di particolare potenza per eseguire le applicazioni,
- possibilità di accedere al sistema in qualsiasi momento, anche in viaggio, basterà la connessione a internet,

- i dati sono completamente gestiti dal service provider che ne garantisce il corretto backup, questo evita la perdita di dati legata a un possibile guasto del nostro hard disk.

Nonostante questi innegabili vantaggi ci sono alcuni aspetti negativi da prendere in considerazione, per alcuni dei quali però già esiste una soluzione:

- il miglioramento della qualità e della velocità di connessione alla rete (banda larga) sono il primo dei fattori da cui non si può prescindere e che permettono di utilizzare questi servizi. In Italia questo potrebbe essere un problema. Fortunatamente a tal proposito sono state sviluppate nuove tecniche di accesso e rappresentazione dei dati che risentono meno dell'inevitabile sbilanciamento della velocità di banda tra l'utente e l'esterno, a titolo di esempio Ajax, un insieme di tecniche di programmazione usate per creare applicazioni web asincrone,

- senza connessione di rete, un'applicazione cloud non può funzionare a meno di meccanismi di caching previsti dal browser come ad esempio WebStorage di HTML5,

- l'autenticazione e l'autorizzazione sono realizzate su una rete che non sicura quale internet, che può essere manomessa e monitorata. L'utilizzo di soluzioni di cifratura è una soluzione d'obbligo per proteggere i dati che transitano dal service provider all'utente finale,

- in caso di guasto al server, i servizi potrebbero essere impossibili da raggiungere.

I servizi offerti dal cloud computing sono molteplici, si va dalla semplice gestione di archivi di documenti, testi, foto e video, alla gestione di posta elettronica, agenda appuntamenti, etc. Spesso questi servizi possono essere composti in base alle necessità dell'utente in modo che egli abbia strumento personalizzato fatto di un collage di funzionalità derivanti dalla somma di singoli servizi web. I maggiori players di questa nuova sfida sono Microsoft e Google. Vediamo solo alcune tra le numerosissime proposte:

Microsoft offre una serie di servizi software online gratuiti che estendono le potenzialità del sistema operativo Windows, rendendo possibile l'accesso via web a numerose funzionalità che si arricchiscono ogni giorno. L'obiettivo finale è quello di una migrazione di Windows verso servizi web e modelli.

Il nuovo servizio di Apple, iCloud, permette di accedere a musica, foto e documenti direttamente dal telefono cellulare o dall'iPad,

consentendo di sincronizzare contatti, documenti, email, applicazioni, impostazioni, immagini e altro tramite wifi, in modo del tutto automatico e tra tutti i dispositivi.

Google Apps, pensata per le aziende, riorganizza tutti i servizi di Google (Gmail, Google Docs, etc.) sotto un unico gruppo: posta elettronica, rubrica, calendario, suite per l'ufficio e storage di documenti accessibili da qualsiasi pc, telefonino, smartphone dotato di connessione ad internet.

Altro servizio offerto da Google è Google Book Search, uno strumento sviluppato per consentire la ricerca di libri antichi digitalizzati o in commercio, si tratta di un'offerta interessante ma allo stesso tempo complessa e articolata per le implicazioni che comporta. Nasce infatti il problema della possibile violazione dei diritti d'autore. In questo senso Google si propone di offrire uno strumento di promozione del libro e non come violatore di una proprietà intellettuale.

1.6 Conclusioni

In questo contesto di applicazioni distribuite eterogenee, si è sentita finora la mancanza di un linguaggio prettamente orientato al web, in grado di espletare le funzionalità sia lato client che lato server.

Javascript è un linguaggio di scripting ideato specificatamente per il lato client e, anche se impiegato per funzionalità lato server, il suo utilizzo risulta complesso e limitato a determinate funzioni di nicchia.

Il linguaggio di nuova concezione Dart sopperisce a questa mancanza, fornendo al contempo un linguaggio web oriented, dotato di supporto per la programmazione sia lato client che lato server; un linguaggio strutturato secondo il paradigma object oriented, che supporta tutte le caratteristiche collegate a questo paradigma, come ereditarietà, polimorfismo e riuso in generale; un linguaggio di scripting di facile comprensione e pensato per fornire un buon compromesso tra stabilità e velocità di esecuzione.

Dart si presta efficacemente quindi allo sviluppo di applicazioni particolarmente complesse lato server come di applicazioni lightweight lato client, in grado di collocarsi perfettamente nello scenario presentato precedentemente.

Capitolo 2

Html 5

2.1 Introduzione

HTML è uno dei linguaggi fondamentali per strutturare e presentare contenuti all'interno del World Wide Web.

Lo scenario di Internet è molto cambiato negli ultimi anni, in passato il Web era fortemente collegato al concetto di ipertesto e l'attività più usuale per gli utenti era quella di fruire contenuti, tipicamente in forma testuale. Le applicazioni web erano scarse, costose sia in termini di sviluppo che di necessità di banda, ciò era dovuto in gran parte alla bassa velocità di connessione e al limitato investimento sul media.

Questo era ottimamente rappresentato da HTML, un linguaggio essenzialmente orientato a facilitare la stesura di semplici documenti testuali fra loro collegati.

Negli anni seguenti l'attenzione verso la rete ha subito una forte impennata, invogliando di conseguenza investimenti e ricerca, ciò ha influito positivamente sia per quanto riguarda la diffusione che la velocità di connessione alla rete stessa. Al modello precedente, dove l'utente si limitava alla fruizione di contenuti, si aggiunge ora la possibilità di essere esso stesso creatore attraverso applicazioni web sempre più sofisticate ed interessanti.

Tale livello di complessità, si è però trovato di fronte a un insieme di specifiche non adatte a questi scopi e che quindi si sono prestate al compito solo a scapito di numerosi hack e workaround. Possiamo trovare ovunque esempi di questo utilizzo 'non intenzionale', ad esempio il caso degli attributi *rel* e *ref* che hanno assunto nel tempo valori non previsti, (es.: nofollow) anche esterni alla loro funzione naturale (es.: l'utilizzo di questi due attributi in librerie come Lightbox).

Allo stesso tempo, la crescita del web ha evidenziato delle strutture di contenuto ricorrenti che sono messe in evidenza in modo particolare dai blog: testata, menu di navigazione, articoli, testo a pie' di pagina,

etc. Anche la parte che concerne il singolo articolo presenta di norma lo stesso insieme di informazioni, come per esempio: l'autore, la data di pubblicazione, il titolo e il corpo del messaggio.

Con HTML 4, gli sviluppatori web sono costretti ad utilizzare strutture anonime, come $< div >$ e $< p >$,aggiungendo un valore semantico con l'utilizzo di attributi quali *class* e *id*, questo perchè, HTML 4, non è in grado di fornire gli strumenti appropriati a ottenere una valida gestione e classificazione del contenuto.

L'HTML 5 nasce per risolvere tali problemi proponendo agli sviluppatori un linguaggio capace di adattarsi alle nuove necessità, sia punto di vista della strutturazione del contenuto sia da quello dello sviluppo di vere e proprie applicazioni.

HTML 5 è una evoluzione dell'HTML 4 e dell'XHTML 1.1 regolamentata da un insieme di specifiche gestite dal W3C (World Wide Web Consortium) e dall'apposito gruppo di lavoro WHATWG (Web Hypertext Application Technology Working Group).

Il progetto è stato annunciato nel 2007, mentre la prima bozza è stata pubblicata nel 2008. L'ultima versione della bozza di HTML 5 è datata 23 gennaio 2012 ma le specifiche non saranno completate prima del 2014. Molte delle porzioni della bozza potranno tuttavia venire rilasciate in corso d'opera, senza attendere una certificazione definitiva.

2.2 Novità di HTML5

HTML 5 propone una serie di innovazioni, che hanno l'obiettivo di soddisfare tutte quelle esigenze di interazione e multimedialità di cui necessita una qualunque applicazione Web 2.0. Alcune di queste funzionalità possono essere ad esempio:

- permettere all'utente di salvare informazioni sul device;
- accedere all'applicazione anche in una momentanea assenza di connessione;
- gestire flussi multimediali;
- accedere e manipolare informazioni generate in tempo reale dall'utente attraverso microfono o webcam.
- gestire lo storico della navigazione;
- generare grafica 2D e 3D in tempo reale;

- comunicare in modo bidirezionale con il server e con altre applicazioni;
- eseguire operazioni in background;
- editare contenuti complessi;

2.3 Obiettivi

L'obiettivo principale quello di migliorare il linguaggio fornendo supporti ai più recenti dati multimediali mantenendo allo stesso tempo una buona comprensibilità sia per l'uomo che per le macchine e per tutti i tipi di programmi con cui possa venire in contatto, come browser web o parser. In particolare quello che si vuole ottenere :

1. fornire agli sviluppatori di siti web strumenti in grado di stare al passo con il Web 2.0, il quale è costituito da vere e proprie applicazioni basate sul browser e di forte interazione con l'utente.
2. includere istruzioni per correggere errori e per migliorare l'interoperabilità rimediando così alle molteplici disfunzioni delle versioni precedenti di HTML. Nelle intenzioni dei promotori, HTML 5 dovrebbe sostituire HTML 4 ma anche XHTML1, DOM2HTML e Javascript.
3. rendendere l'HTML più facile da scrivere evitando una normativa troppo inflessibile, come quella proposta dall'XHTML 2 raccogliendo però la sua eredità semantica.

Il risultato raggiunto da questo insieme di intenti è piuttosto articolato e complesso: assistiamo innanzitutto a un cambiamento del modello di markup che vede modificate, in modo rilevante, le basi della propria sintassi e le regole per la disposizione dei contenuti nella pagina; a questo segue un incremento delle API Javascript, estese per essere in grado di supportare tutte le funzionalità richiese da un'applicazione web moderna.

La specifica HTML 5 prevede nativamente la presenza di API e di un DOM; introduce inoltre una specifica sul processamento dei documenti, rendendo possibile quindi una gestione standard degli errori da parte di browser differenti o agenti software eterogenei.

HTML 5 mira a disaccoppiare il contenuto di una pagina web dalla struttura (definita tramite il markup) e dall'aspetto (definito dai fogli di stile).

Inoltre, HTML 5 prevede il supporto per la memorizzazione locale di grosse quantità di dati scaricate dal browser e la nascita di elementi specifici per il multimedia, di cui Internet è molto ricca.

In HTML 5 confluiscono altre tecnologie oltre all'HTML inteso, in senso stretto, come semplice linguaggio di markup, che abbracciano diverse componenti dello sviluppo per il Web. Il W3C ci dùna panoramica degli standard e delle tecnologie coinvolte:

- *Semantica*: il Web viene reso più fruibile agli utenti e ai programmi grazie a un insieme più corposo di tag, all'utilizzo di RDF (Resource Description Framework), ai microdati e ai microformati i quali permettono di dare un senso alla struttura in chiave semantica. I microformati consentono alle pagine la possibilità di veicolare delle micro informazioni per diversi generi di contenuto: contatti, relazioni, eventi, prodotti, discussioni, ricette, ecc. Questa non è una novità di HTML5 poichè utilizzano sia elementi come ($< span >, < div >, < a >$ ecc.) che attributi come ($rel, class, rev$, ecc.) che sono compatibili con le precedenti versioni di HTML. Nonostante non siano propri di HTML 5 vengono comunque inclusi tra le tecnologie correlate perchè aggiungono un significato semantico al contenuto. I microformati non sono invasivi sull'output generato e sono fruibili sia dall'utente, tipicamete mediante appositi plugin del browser, sia da sistemi di parsing automatico.

- *Connettività*: I WebSockets ci portano a una connettività sempre più efficiente, il chè consente di effettuare chat in tempo reale, utilizzare giochi più veloci e migliorare la comunicazione tra client e server;

- *Offline e gestione dei dati*: grazie a tutte le nuovi API per la gestione dei file, LocalStorage, App Cache, IndexedDB, le applicazioni sono in grado di avviarsi in modo molto più veloce e di funzionare anche in una temporanea assenza di connettività;

- *Accesso al dispositivo*: le applicazioni sono in grado di accedere alle caratteristiche del dispositivo, come ad esempio webcam e microfoni, ai dati locali, come contatti e calendario, e a caratteristiche come inclinazione e orientamento, tutto questo a partire dalle API per la geolocation;

- *Multimedia*: in HTML5 la riproduzione di contenuti multimediali, come audio e video, è gestita nativamente, senza la necessità di ricorrere a plugin esterni;

- *Grafica*: con le nuove funzionalità di HTML 5 (canvas) e altri standard già esistenti (SVG, CSS3) si possono realizzare effetti grafici veramente interessanti. Molti programmi, sia commerciali che open source, supportano SVG (ad esempio Visio, OpenOffice Draw, Google Docs, Corel Draw, GIMP, Inkscape, etc) per cui è facile inserire questo tipo di contenuto nelle pagine.

- *CSS3*: presenta numerose novità, le cui specifiche sono gestite separatamente dal W3C, in moduli diversi a seconda delle caratteristiche: selettori, posizionamento, box model, gestione del testo, dei font, dei bordi, degli sfondi, dei colori, media query, trasformazioni, animazioni, ecc. La vasta scelta tra effetti, come trasparenze, ombreggiature, immagini di sfondo multiple, e stili diversi che si possono applicare ai contenuti senza dover adattare il markup, consentono una personalizzazione grafica straordinaria.

- *Prestazioni e integrazione*: creare applicazioni con prestazioni elevate e contenuti dinamici velocissimi diventa più semplice grazie all'impiego di Web Workers e XMLHttpRequest 2;

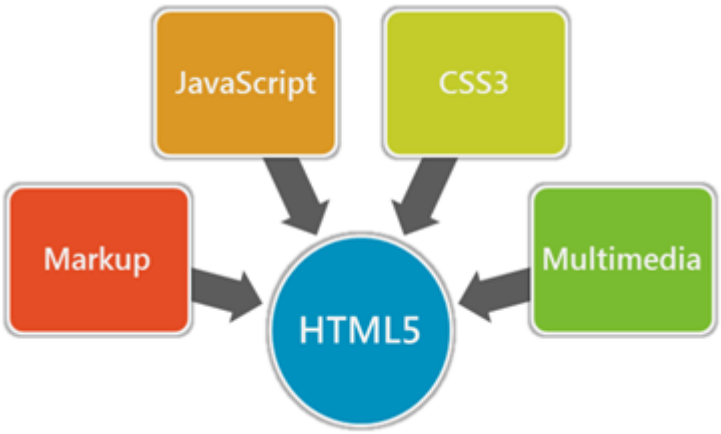


Fig.1 HTML5 può essere visto come la combinazione di markup, Javascript, CSS3 e multimedia

2.4 Sintassi

La sintassi costituisce la maggiore differenza tra le attuali versioni di HTML e XHTML. HTML, il linguaggio creato da Tim Berners-Lee che tuttora è nei nostri browser è stato studiato come applicazine di SGML - Standard Generalized Markup Language. Ne è la prova la dichiarazione di Document Definition Type posta nella prima riga di una pagina Web ad indicare la grammatica HTML usata nel documento:

$$<!DOCTYPE HTML PUBLIC'' - //W3C//DTD HTML 4.01 //EN''''http://www.w3.org/TR/html4/strict.dtd'' >$$

In realtà la quasi totalità dei browser ignora la definizione e interpreta il documento secondo logiche più permissive, frutto di anni di eccezioni e di esperienza accumulata su pagine malformate.

L'XHTML, invece, è una versione della sintassi HTML costretta all'interno delle regole XML, a sua volta grammatica SGML: questo approccio dovrebbe implicare un maggior rigore nella pagina e l'aderenza a regole quali l'obbligo di chiusura di tutti i tag. Il parser XML inoltre dovrebbe sospendere l'interpretazione della pagina al primo errore rilevato.

In tale scenario HTML5 introduce una importante novità, l'obiettivo delle specifiche è difatti quello di definire un linguaggio che possa poi essere implementato su entrambe le sintassi, rompendo in modo definitivo il legame tra HTML e SGML, formalizzando e traducendo in standard le regole che da tempo sono state adottate nei browser.

Per indicare un documento HTML5 è nata la seguente semplice istruzione:

```
<!DOCTYPE html>
```

Che si affianca a quella da utilizzare in caso si intenda scrivere una pagina XHTML5:

$$< html xmlns =" http://www.w3.org/1999/xhtml" xml : lang =" en" >$$

Da osservare che la versione di HTML non è più contenuta nel doctype come nelle versioni precedenti ma, poichè l'HTML 5 deve supportare anche vecchie specifiche, verrà utilizzato per documenti HTML 4.01 o XHTML 1.0 esistenti, inoltre, ogni futura versione di HTML a sua volta dovrà supportare specifiche HTML 5.

Questo processo porta a non dare più importanza allo scrivere esplicitamente il numero della versione dell'HTML. D'ora in poi passerà il concetto che i browser devono poter interpretare le caratteri-

stiche e le specifiche del linguaggio e non il doctype che quindi potrà essere semplice da leggere, da capire e soprattutto da ricordare.

La sintassi HTML 5 è caratterizzato da una spiccata flessibilità e semplicità di implementazione. Gli elementi si possono dividere in tre categorie in base alla tipologia di tag da usare per implementarli.

- *Elementi normali*: sono quegli elementi che racchiudono dei contenuti sotto forma di testo, commenti HTML, altri elementi HTML, etc. Esempi di questo tipo di elemento sono i paragrafi ($< p >$), le liste ($< ul >$), i titoli ($< h1 >$), etc. Salvo casi particolari, gli elementi normali vengono definiti attraverso un tag di apertura ($< p >$) e un tag di chiusura ($< /p >$).

- *Elementi vuoti*: sono quelli che non possono avere un contenuto, per tali elementi viene utilizzato un tag vuoto. Alcuni esempi: area, base, br, col, command, embed, hr, img, input, keygen, link, meta, param, source, track, wbr. Per questo tipo di elementi la chiusura del tag è opzionale, a differenza di quanto avviene in XHTML dove invece la chiusura del tag è obbligatoria. Possiamo dunque definire un tag $< img >$ secondo le regole XHTML:

  ```
  <img src="immagine.png" alt="testo" />
  ```

  O seguendo la vecchie regole di HTML 4:

  ```
  <img src="immagine.png" alt="testo">
  ```

- *Elementi provenienti da altri namespace*: per questi elementi sono richiesti i tag 'autochiudenti'. Si tratta degli elementi adottati da specifiche esterne, come SVG e MathML.

A differenza del più restrittivo XHTML, che riconosce solamente tag definiti con lettere minuscole, in HTML5 i tag possono essere definiti sia con lettere minuscole che maiuscole.

In alcuni casi particolari, inoltre, quando il browser è in grado di determinare ugualmente il contenuto del tag, può essere omesso il tag di apertura o quello di chiusura; alcuni esempi sono i tag $< head >, < body >$ e $< html >$, che in alcuni casi possono essere del tutto omessi.

Per quanto riguarda gli attributi HTML5 ha una filosofia meno rigida rispetto a quello di XHTML, rendendo non obbligatorio il fatto di racchiudere gli attributi tra apici o virgolette:

- *Attributi vuoti*: nel caso in cui non sia necessario specificare un valore per l'attributo, basta indicarne il nome; il valore verrà poi automaticamente associato alla stringa vuota. Per esempio: Secondo le regole XHTML:

  ```
  <input checked="checked" />
  ```

  In HTML 5:

  ```
  <input checked>
  ```

- *Attributi senza virgolette o apici*: è possibile definire il valore di un attributo senza racchiuderlo tra virgolette o tra apici. Per esempio:

  ```
  <div class="titolo">
  <div class='titolo'>
  ```

  potrebbero diventare semplicemente:

  ```
  <div class=titolo>
  ```

Anche se HTML5 permette un gran numero di omissioni o semplificazioni, per una maggiore leggibilità, è comunque consigliabile rappresentare tutti i tag necessari in forma completa.

Un'altro tentativo che mette in atto HTML 5 è quello di eliminare tutto ciò che si è rivelato scarsamente utile o dannoso ma che ancora popolava HTML 4. Le specifiche sanciscono definitivamente la fine di tutta una serie di elementi e attributi che vengono mantenuti solo formalmente per preservare la retrocompatibilità ma sono espressamente vietati nella creazione di nuovi documenti.

Gli appartenenti a questa famiglia sono tutti quei tag che appartenevano alla parte di presentazione ma che sono caduti i disuso dopo l'introduzione dei fogli di stile:

$< basefont >, < big >, < center >, < font >, < s >, < strike >$
$< tt >, < u >$

A questi si aggiungono attributi come:

$< align >, < valign >, < background >, < cellspacing >$
$< bgcolor >$, e $< border >$

Vengono eliminati anche tag già obsoleti in HTML 4 come:

$< acronym >, < applet >, < isindex >, < dir >$

e tutti i quanti i tag legati al concetto dei frame, già ampiamente demonizzati in precedenza, come:

$< frame >, < framset >< noframes >$.

2.5 Struttura

Ripercorrendo la storia del World Wide Web possiamo osservare come nel tempo sia cambiato il modo di gestire l'impaginazione dei contenuti, passando da strutture basate su tabelle a layout organizzati in modo gerarchico attraverso l'uso di $< div >$, semplificando così in modo significativo la struttura del DOM e delegando ai fogli di stile il compito di gestire la visualizzazione e l'impaginazione.

Tipicamente le pagine presentano alcuni elementi comuni: una testata, uno o più menù di navigazione, un contenuto principale, un piè di pagina e così via.

In una pagina HTML 4.01 questi elementi sono semplicemente $< div >$, con una certa classe o un ID per gestirne posizionamento, dimensioni e formattazione, ma sono di fatto formalmente indistinguibili, ad esempio per un motore di ricerca.

HTML5 introduce nuovi tag per attribuire un significato semantico ai diversi blocchi che compongono la pagina:

$< header >, < article >, < nav >, < footer >, < aside >$
$< section >, < figure >$ ecc.

Per lasciare al markup esclusivamente il compito di definire il contenuto sono stati soppressi alcuni tag di formattazione come:

$< big >, < center >, < font > o < strike >$

l'aspetto della pagina deve essere controllato via CSS.

2.5.1 Regole per la strutturazione del testo

Ogni pagina HTML diventa difficile da processare a causa della mancanza di una struttura. Si deve analizzare la struttura di ogni sezione, facendo attenzione ai livelli di intestazione.

HTML5 introduce nuovi tag che risolvono questo problema suddividendo il documento in sezioni logiche:

- $< header >$: l'elemento header rappresenta una sezione della pagina contenente un'introduzione e una serie di elementi che aiutano la navigazione (es.: menù di navigazione) e che mostrano contenuti introduttivi all'argomento della pagina o di una sezione (es.: Titolo, Sottotitolo, etc.). Un elemento header di norma contiene le intestazioni per la sezione, ma può contenere altri elementi, come un sommario, un campo di ricerca, etc. È da notare che l'elemento header viene posto dentro una sezione e non come sua introduzione. Il tag header introdotto dall'HTML5 si differenzia dal tag head già presente nelle precedenti versioni in quanto l'head è unico in una pagina mentre è possibile avere più di un header ad esempio un tag header potrebbe essere inserito in una section per definire la parte iniziale di quella sezione specifica. Per definizione, infatti, una section è un raggruppamento tematico di contenuti, generalmente indicato da un titolo.

- $< footer >$: l'elemento footer appresenta l'elemento conclusivo della sezione a cui si riferisce, solitamente contiene informazioni riguardo la sezione in cui è contenuto; ad esempio informazioni sull'autore del contenuto, link a documenti collegati, informazioni su copyright ecc. Gli elementi footer non devono essere posti obbligatoriamente in fondo al documento, sebbene sia una pratica comune. Questo nuovo tag rivoluziona un po' il modello che i web designer, in HTML5, infatti, possiamo avere più footer, uno per ogni contenuto tematico della pagina.

- $< nav >$: l'elemento nav racchiude una serie di link ad altre pagine interne o esterne al sito. Si può anche usare per indicare i sottomenu e gli eventuali link per scorrere gli articoli (avanti e indietro); non tutti i gruppi di link devono essere posti in un elemento nav: solo i link principali sono appropriati.

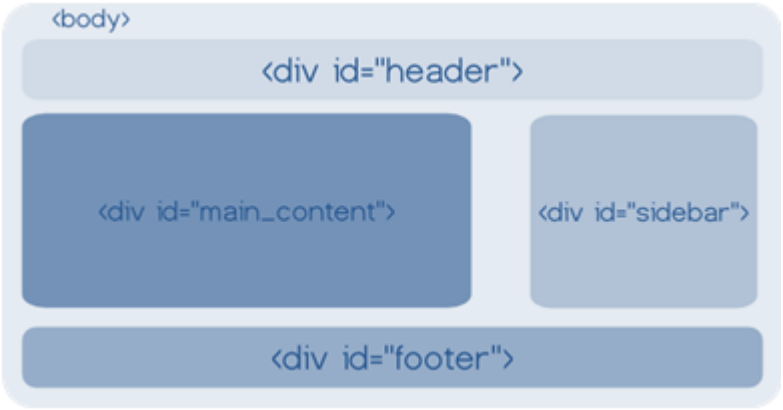
- $< section >$: l'elemento section rappresenta una generica sezione di un documento ed è usato per raggruppare contenuti tematicamente relazionati. Sembra assomigliare ad un elemento $< div >$ che viene spesso utilizzato come contenitore generico ma, mentre il *div* non ha un significato semanticamente rilevante, il tag *section* è usato esplicitamente per raggruppare contenuti che hanno relazioni tra loro. È possibile sostituire i *div* con il

tag *section* sempre ricordando, però, di chiedersi prima di tutto se gli elementi che state raggruppando sono relazionati tra loro.

- $< article >$: l'elemento article accoglie il contenuto principale della pagina, che per i motori di ricerca è fra le cose più importanti; rappresenta una sezione della pagina che consiste in una entità indipendente. Potrebbe essere un post di un forum o di un blog, un articolo di giornale, un commento da parte di un utente o un altro contenuto che possa avere significato da solo anche estratto dal contesto. È possibile avere più livelli di elementi *article*, in questo caso gli elementi più interni sono comunque legati a quello esterno, pur essendo indipendenti. Ad esempio, i commenti ad un post di un blog, possono essere resi come article all'interno dell'elemento article che rappresenta il post. Possiamo affermare che *article* è una specializzazione di *section* utilizzato per contenuti relazionati 'autosufficienti': in altre parole contenuti che hanno relazione tra loro ma vivono anche indipendentemente gli uni dagli altri.

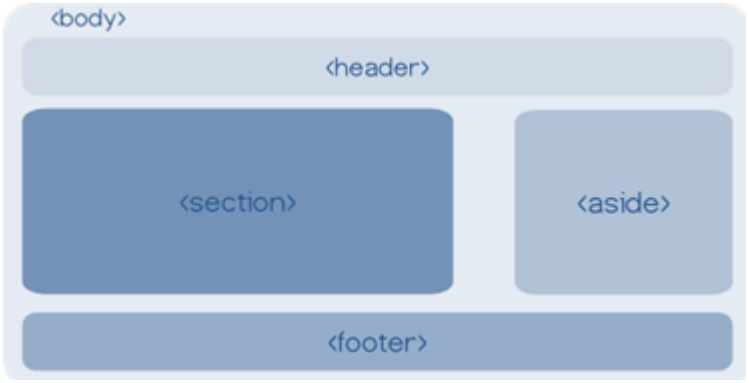
- $< aside >$: l'elemento aside è un tag per i contenuti correlati, serve per specificare altri articoli simili all'argomento. Solitamente viene resa come una colonna posta a lato del contenuto principale, ma nulla vieta che possa trovarsi all'interno. Un esempio pratico in cui è ottimo utilizzare questo tag sono le citazioni: in generale parliamo di contenuti interessanti da leggere ma che possono anche essere eliminati senza cambiare la comprensione del contenuto. Ricordiamo che contenuti che, per esigenza di design, devono apparire nella sidebar non è detto che debbano essere inseriti in un tag aside. Per esempio le informazioni relative alla vita dell'autore di un articolo che dobbiamo inserire in una sidebar vanno inserite in un tag footer e non in un aside.

Vediamo ora un esempio delle novità introdotte con HTML 5 confrontando con quello che avremmo avuto con HTML 4.01. Si tratta di un documento con un header, un footer, una colonna con il contenuto principale e una barra laterale. Se avessimo dovuto realizzarlo in HTML 4.01 avremmo avuto qualcosa del genere:




```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN
              http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>TITOLO</title>
    <meta http-equiv="Content-Type"content="text/html;charset=utf-8">
  </head>
  <body>
    <div id="header">
      <h1>Titolo</h1>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">Blog</a></li>
        <li><a href="#">Autori</a></li>
      </ul>
    </div>
    <div id="main\_content">
      <h2>Titolo contenuto principale</h2>
      <p>Contenuto principale</p>
    </div>
    <div id="sidebar">
      <p>altro contenuto</p>
    </div>
    <div id="footer">
      <p>informazioni varie</p>
    </div>
  </body>
</html>
```

Sfruttando i nuovi elementi, si può trasformare questo layout in un documento molto più valido dal punto di vista semantico:



Che tradotto in codice diventa:

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>HTML5</title>
  </head>
  <body>
    <header>
      <h1>Titolo</h1>
    </header>
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">Blog</a></li>
        <li><a href="#">Autori</a></li>
     </ul>
    </nav>
    <section>
      <h1>Titolo per il contenuto principale</h1>
      <p>Contenuto principale del sito.</p>
    </section>
    <aside>
      <p>altro contenuto</p>
    </aside>
    <footer>
      <p>Informazioni varie</p>
    </footer>
  </body>
</html>
```

Dunque sono stati sostituiti:

`<div id="header"> con <header>`

`<div id="sidebar"> con <aside>`

`<div id="main_content"> con <section>`

`<div id="footer"> con <footer>`

La questione però va oltre delle semplici sostituzioni, gli elementi introdotti in HTML5 non sono dei semplici 'sostituti semantici' dei *div*. La ragione pratica è associata alla struttura dei titoli della pagina: osservando il codice con attenzione si può notare che, mentre nel primo esempio il titolo del contenuto principale è stato reso con un $< h2 >$, nel secondo è stato utilizzato un$< h1 >$. Questo perchè in HTML 5 è stato introdotto un nuovo algoritmo per la gestione dei titoli che, in pratica, permette di utilizzare all'interno di ogni elemento $< section >$ tutti i titoli (da h1 ad h6).

I vantaggi rispetto alla normale gestione di HTML 4 sono dovuti al fatto che così si possono gestire layout più complicati rispetto ad un sito web lineare garantendo una struttura sempre significativa. Le ragioni per utilizzare markup semantico possono sembrare molto deboli dal punto di vista umano, ma per le macchine (motori di ricerca, screen reader, ecc) è impossibile distinguere un paragrafo da un'immagine o altro.

Vediamo come si potrebbe strutturare un blog, utilizzando i nuovi elementi: i post di un blog sono entità indipendenti dal contenuto del sito, quindi possiamo utilizzare l'elemento $< article >$. Un post è formato dai seguenti elementi:

- titolo e data di pubblicazione: possiamo inserire queste informazioni in un elemento header poichè introducono il contenuto.
- contenuto del post: utilizzeremo tutti gli elementi già noti di HTML4.
- metadati del post: quindi categorie, tag, nome autore, link a commenti. Si potrebbe pensare di utilizzare l'elemento $< aside >$ però queste informazioni sono relative al contenuto, dunque sfrutteremo l'elemento $< footer >$.
- altri elementi: link ad articoli correlati, icone di bookmark, link per feed rss, possono essere racchiusi da un $< aside >$.
- commenti: le specifiche dicono esplicitamente che un commento di un post rappresenta un'entità a sè stante, anche se collegata al post. Dunque potremmo inserirli in elementi $< article >$.

Sottolineiamo il fatto che non è necessario che l'elemento $< footer >$ debba trovarsi in fondo alla pagina, l'elemento $< aside >$ a lato e così via: bisogna pensare al significato del contenuto e non alla sua posizione all'interno della pagina.

2.6 Canvas e multimedia tag

Altra novità sono gli elementi $< canvas >, < video > e < audio >$, che rimpiazzano l'uso del generico $< object >$.

Questi elementi sono stati introdotti per rendere semplice incorporare elementi multimediali senza dover ricorrere all'utilizzo di API o plugin esterni magari anche proprietari.

2.6.1 Canvas

Può essere impiegato per disegnare ed operare con elementi grafici. Il tag $< canvas >$ necessita di un linguaggio di scripting di supporto, come Javascript, per funzionare correttamente e svolgere a pieno le sue potenzialità.

Utilizzando questo elemento è possibile costruire immagini bitmap attraverso tag HTML e Javascript. Il suo utilizzo è piuttosto semplice e non differisce affatto dagli altri tag del linguaggio HTML.

Formalmente è un semplice contenitore e, come tale, dispone di un tag di apertura ($< canvas >$) ed uno di chiusura ($< /canvas >$). Se non vengono indicate esplicitamente le dimensioni (tramite gli attributi width ed height) la dimensione assegnata al contenitore è quella di default, un rettangolo con base 300 e altezza 150.

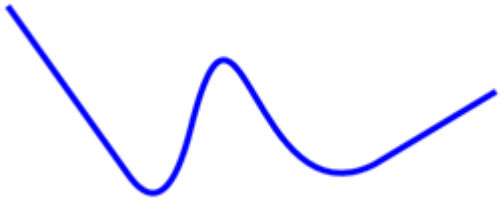
L'attributo id, ovviamente, non è indispensabile ma è buona norma definirlo sempre per avere un riferimento univoco per ogni oggetto che si sta impiegando nella pagina.

Attualmente è supportato dalle versione correnti di Mozilla Firefox, Google Chrome, Internet Explorer, Safari e Opera. L'elemento introduce una regione rettangolare, definita all'interno del codice HTML, che mette a disposizione del codice Javascript un set molto corposo di API per il disegno, permettendo la generazione dinamica di elementi 2D: grafici, animazioni, giochi, composizione di immagini, ecc.

Vediamo un semplice esempio:

```
<!doctype html>
<html>
 <head>
  <meta charset="UTF-32">
  <title>Canvas</title>
  <header>
    <h2>CANVAS</h2>
    <p>28 marzo 2012</p>
   </header>
  <script>
  window.onload = function(){
    var canvas = document.getElementById("canvas-esempio");
    if (canvas.getContext) {
      var context = canvas.getContext("2d");
      context.beginPath();
      context.moveTo(100, 20);
      context.lineTo(200, 160);  // Prima linea
      context.quadraticCurveTo(230, 200, 250, 120);// Curva
      context.bezierCurveTo(290,-40, 300, 200, 400, 150);
      context.lineTo(500, 90);  // Seconda linea
      context.lineWidth = 5;
      context.strokeStyle = "#0000ff";
      context.stroke();
    }
  };
  </script>
 </head>
 <body>
   <canvas id="canvas-esempio" width="600"
    height="1580"></canvas>
 </body>
</html>
```

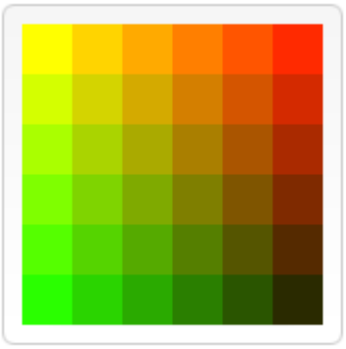
Risultato:



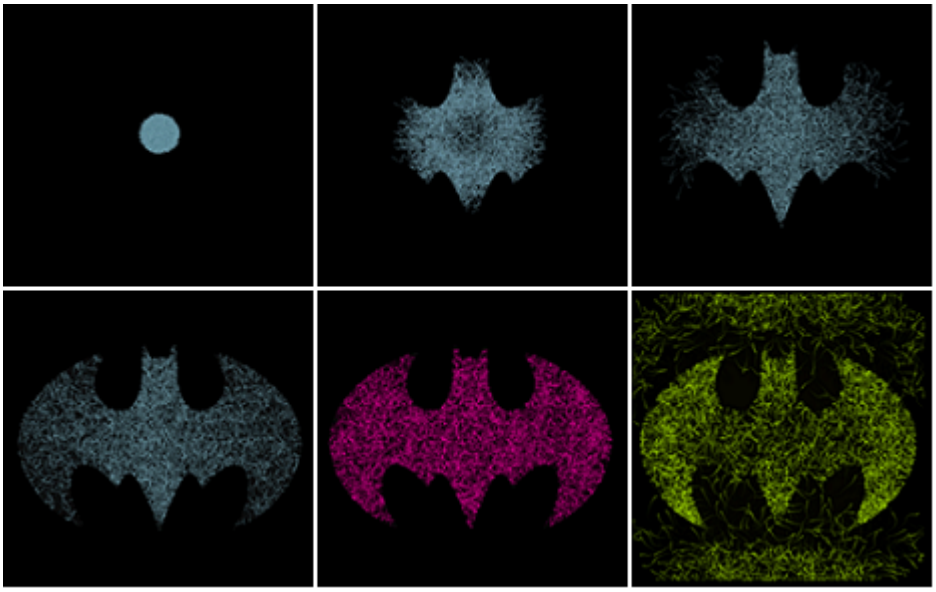
Altro esempio:

```
<!doctype html>
     <html>
     <head>
      <script type="application/Javascript">
        function draw() {
   var ctx = document.getElementById('canvas').getContext('2d');
   for (i=0;i<6;i++){
     for (j=0;j<6;j++){
       ctx.fillStyle = 'rgb(' + Math.floor(255-42.5*i) + ',' +
                       Math.floor(255-42.5*j) + ',0)';
       ctx.fillRect(j*25,i*25,25,25);
     }
   }
}
      </script>
     </head>
     <body onload="draw();">
       <canvas id="canvas" width="150" height="150"></canvas>
     </body>
    </html>
```

Risultato:



Un ulteriore esempio di quello che si può ottenere con l'elemento canvas e HTML5. Si tratta in sostanza di utilizzare canvas per replicare, attraverso tanti piccoli filamenti colorati su sfondo nero, l'immagine che gli viene passata. L'effetto è davvero spettacolare.



2.6.2 Video e audio

In HTML5 abbiamo a disposizione due nuovi tag molto simili nell'utilizzo: audio e video che ci danno la possibilità di riprodurre contenuti multimediali senza far ricorso a plug-in esterni come Flash o Silverlight.

Entrambi i tag producono una interfaccia in quanto dispongono di una barra di controllo per agire sul contenuto in ascolto o in visualizzazione. Questi tag vanno inseriti all'interno della pagina, nel punto in cui lo riteniamo opportuno, e rientrano nel DOM della stessa, interagendo e influenzando gli altri elementi presenti. Possiamo inserire i due tag come segue:

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML5: audio e video</title>
  </head>
```

```
  <body>
    <h1>Audio</h1>
    <audio src="test.mp3" controls />

    <h1>Video</h1>
    <video src="test.mp4" controls>
      Il tuo browser non supporta i video
    </video>
  </body>
</html>
```

Attraverso l'attributo *src* si può indicare l'url del contenuto multimediale da riprodurre. L'attributo *controls* indica di visualizzare i controlli standard offerti dal browser per la classica gestione di play, pause, seek e volume, di cui ogni contenuto multimediale normalmente necessita.

Per il tag audio la mancata indicazione dell'attributo controls comporta l'avvio della traccia audio, mentre per il tag video la riproduzione non risulta controllabile, se non sfruttando il menu contestuale del browser.

Da notare, infine, che all'interno del tag possiamo indicare il markup da visualizzare nel caso il browser non dovesse supportare HTML5. Nel caso in cui il browser non supporti il tag video, viene caricato automaticamente il player Silverlight e sarà lui ad effettuare la riproduzione.

Possiamo utilizzare altri attributi per controllare la riproduzione del contenuto:

- *autoplay*, per avviare automaticamente il contenuto non appena la pagina si avvia
- *loop* ripete all'infinito il contenuto e può essere utile ad esempio per creare un suono di sottofondo continuo per la pagina:

  $< audiosrc =" test.mp3" autoplayloop/ >$
- qualora autoplay non fosse indicato, l'attributo *preload* istruisce il browser su come caricare il contenuto. Il valore predefinito è *auto* e indica che è a discrezione del browser caricare il contenuto in funzione del carico della rete o della previsione di pre-buffering necessario per riprodurlo. Con *metadata* indichiamo che il browser devi limitarsi a caricare i primi byte del file, necessari ad

individuare le caratteristiche degli stream, la durata, la dimensione e il primo frame. Con *none*, infine, il browser capirà che non è necessario caricare immediatamente il file, così da ridurre il traffico da effettuare con il server.

- sul tag *video* abbiamo a disposizione l'attributo *audio* che, se impostato su 'mute', permette di azzerare il volume da applicare alla riproduzione del video.

I formati riproducibili con HTML5 sono oggetto di disputa fra i produttori di browser più diffusi perchè tuttora manca la definizione di uno standard di formato universalmente riconosciuto. Per il video le possibilità sono principalmente MP4 e WebM mentre per l'audio sono MP3 e Ogg Vorbis. In questa fase, abbiamo la possibilità di decidere quale formato supportare, oppure orientarci a supportare tutti i browser, preparando le risorse multimediali in più formati, che i browser scelgono in base a ciò che supportano.

In alternativa all'uso dell'attributo *src*, possiamo usare l'elemento *source*, per definire una o più sorgenti, identificandole in base ai formati specificati: sarà il browser, poi, a decidere di riprodurre quello che è in grado di decodificare.

2.7 Form

I forms sono una risorsa necessaria per richiesta di informazioni, contatto o pareri. Lo sviluppatore nella realizzazione di un form dovrà porre la propria attenzione a due aspetti:

- conformità e validazione dei dati: dovremo controllare ad esempio che nel campo età sia stato inserito un numero positivo e senza decimali, che nel campo sito internet sia stato digitato un indirizzo internet, ecc

- facilitazione dell'inserimento dati: potremmo facilitare l'inserimento di una data con un calendario a comparsa, o la scelte di un colora tramite una tavolozza invece che inserendo un valore numerico

Fin qui ci si è aiutati e ci si aiuta con l'utilizzo di codice Javascript, necessario per permettere di limitare l'invio di form mal compilati o con errori, evitando all'utente l'attesa non necessaria tra il l'invio, l'eventuale fallita validazione dei dati inviati e il conseguente reload della pagina. Ovviamente i controlli di formalità e congruità, oltre

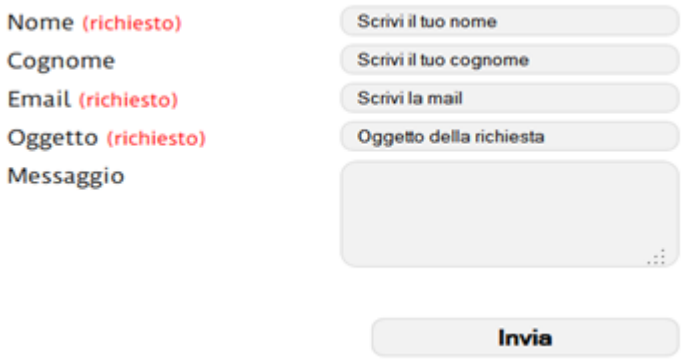
che per mezzo di Javascript lato client, dovrebbero sempre avere un ulteriore controllo lato server.

Le specifiche di HTML5 introducono funzionalità e tecniche che permettono allo sviluppatore da affidarsi unicamente al linguaggio di markup, senza dove ricorrere a Javascript o a plugin esterni. HTML 5 introduce nuove tipologie distinte di 'input type', come ad esempio:

- email,
- number,
- range,
- search

e di diversi nuovi attributi come:

- placeholder: rappresenta un breve suggerimento al fine di favorire l' utente con l' inserimento dei dati nel campo. Tipicamente come suggerimento nel placeholder si inserisce un valore di esempio o una breve descrizione del formato previsto.



- autofocus: è un attributo booleano e serve a impostare il focus su uno specifico elemento del form appena la pagina è caricata. Ad esempio nella home page di Google, appena viene caricata, il focus è automaticamente impostato sul campo per la ricerca

  Esempio di utilizzo dell'attributo:

```
<form action="/" method="get">
  <input type="text" name="myname" id="myid" autofocus>
  <input type="submit" value="Invia">
</form>
```

- required: specifica un campo di testo che deve essere compilato obbligatoriamente, se il campo non viene compilato il form restituirà un errore durante linvio

- min max: I valori min e max descrivono rispettivamente il valore minimo e massimo consentito. Il valore di max deve essere maggiore del valore di min se indicato. Questi attributi si applicano sia alle date sia ai numeri.

2.8 API per Web application

Una delle caratteristiche che hanno determinato il successo del web è l'architettura client-server, una semplice modalità di comunicazione dove c'è un client, che fa interrogazioni al server, e un server che risponde a tali richieste solo quando interrogato. Tale meccanismo è l'ideale per un web statico, come il Web 1.0, costituito da pagine collegate fra loro, in cui la richiesta di un nuovo contenuto viene effettuata dall'utente tramite link. L'architettura client-server non è adatta però ad applicazioni complesse, con comunicazioni bidirezionali e dove le informazioni devono poter arrivare quanto sono disponibili e non solo quando l'utente ne fa richiesta. Queste nuove necessità delle applicazioni web hanno portato allo sviluppo di tecnologie parallele all'HTML, nate allo scopo di fronteggiare il vuoto percepito dall'utente della rete. Sono un esempio di queste tecnologie: Flash, Flex, Google Gears, etc. La necessità di evitare che queste soluzioni estranee al consorzio possano di fatto complicare la gestione dell'intero sistema e la volontà di mantenere il controllo sullo standard hanno portato il gruppo W3C a provvedere in tal senso.

2.8.1 Applicazioni web offline

HTML5 fornisce una nuova funzionalità di caching per supportare le applicazioni web in modalità offline. Può accadere infatti che la connessione alla rete possa non essere momentaneamente disponibile, per far fronte a tale eventualità è stata elaborata una parte della specifica di HTML5 denominata $OfflineWebApplication$ che consente di progettare applicazioni web utilizzabili quando la connessione di rete non è attiva e consiste nella possibilità di far caricare in una cache locale, denominata application cache, dei file di risorse rappresentati da pagine web, filmati Flash, immagini, fogli di stile CSS, Javascript e così via. Il file ha questo formato:

```
CACHE MANIFEST
index.html
      /images/logo.png
      /css/styles.css
      /js/jquery-1.4.min.js
      /js/offline.js
```

Una volta salvato questo file diremo al browser di utilizzarlo specificandolo come attributo del tag html nella nostra pagina, così:

```
<html manifest="offline.manifest">
```

Quando visiteremo il sito in questione il browser chiederà se autorizziamo l'utilizzo della cache e a quel punto i contenuti che abbiamo specificato nel file manifest verranno salvati sul nostro computer così da permetterci una navigazione completa del sito anche quando siamo offline. È importante che il file manifest sia servito dal server web al browser con il MIME type giusto:

```
text/cache-manifest
```

in caso contrario il browser non lo riconoscerà come manifest per la cache. È necessario fare attenzione al fatto che il semplice aggiornamento di un file incluso nel manifest non comporta l'aggiornamento della versione in cache pertanto man mano che andremo ad aggiungere risorse al progetto (immagini, librerie, etc...) sarà necessario includerle anche nel file .manifest.

Il vantaggio principale rispetto alla cache tradizionale è il controllo: mentre solitamente è il browser a decidere quali file memorizzare in cache, con l'application cache possiamo dire noi quali risorse tenere in memoria così da offrire agli utenti un'esperienza di navigazione offline completa.

Inoltre l'application cache consente di salvare file che non sono stati visitati ma che potrebbero essere utili per una navigazione completa del sito. Nella normale cache del browser una pagina per essere memorizzata deve essere stata visitata almeno una volta.

Le offline web applications sono uno strumento molto potente che, combinato con il local storage, consente di creare delle vere e proprie applicazioni indipendenti dalla effettiva connessione ad Internet o meno. L'attuale utilizzo più valido è sui siti web pensati per i telefonini, consentendo agli sviluppatori di creare di fatto delle vere a proprie applicazioni funzionanti anche quando non è presente una connessione di rete.

2.8.2 Memorizzazione di informazioni sul browser

HTML5 fornisce diversi tipi di archiviazione dei dati lato client; tra questi troviamo *LocalStorage* e *SessionStorage*, in genere denominati *WebStorage*.

Il WebStorage permette di gestire la persistenza di elementi su computer client locali in modo simile a quanto è possibile fare con i cookie superandone però le limitazioni e la difficile gestione. Un cookie, pur essendo tuttora il sistema più utilizzato per memorizzare le informazioni, ha diversi svantaggi:

- API Javascript poco pratiche e di difficile utilizzo;
- limitazione di spazio di memorizzazione a soli 4 KB per cookie;
- inefficiente occupazione di banda perchè essi sono trasmessi tra un client e un server a ogni richiesta HTTP;
- problemi di sicurezza nel caso la trasmissione non usi il protocollo HTTPS.

La specifica di HTML 5 introduce quindi due nuovi meccanismi per la memorizzazione:

- *localStorage*: una chiave in questa istanza ha visibilità a livello di dominio, i dati conservati nell'istanza sono quindi visibili in tutte le finestre aperte sul dominio. Local Storage è un'archiviazione persistente
- *sessionStorage*: sessionStorage offre un'archiviazione temporanea. La visibilità delle chiavi appartenenti a questa istanza, è limitata alla finestra del browser in cui questa variabile è stata creata. Una volta chiusa a finestra, i dati contenuti in questa istanza verranno distrutti assieme alla finestra stessa.

  Un'istanza di questo tipo è indicata per casi in cui una transazione è legata ad una singola finestra dell'utente che però può a sua volta avere aperte più finestre del browser con diverse transazioni tutte operative allo stesso tempo. Ad esempio usando sessionStorage sarebbe quindi possibile coordinare l'apertura contemporanea di due distinti account GMail sullo stesso browser.

L'utilizzo è molto semplice, si usano i metodi *setItem* e *removeItem* per inserire e cancellare item utilizzando un meccanismo chiave valore. Ad esempio per inserire un valore:

$localStorage.setItem("keyName","keyValue")$;

per rimuoverlo:

$localStorage.removeItem("keyName")$;

Attualmente è supportata dai seguenti browser:

Browser	Dalla versione
Internet Explorer	8
Firefox	3.5
Chrome	4
Safari	4
Opera	10.5
iPhone	2.0
Android	2.0

Rispetto ai cookies il Web Storage è la soluzione ideale per progetti in cui abbiamo bisogno di salvare lo stato di un'applicazione e i dati trasmessi sono pesanti: il limite di 5 Mb consente di avere un database piuttosto ampio; nel caso in cui dovessimo superare i limite al nostro tentativo di aggiungere altri dati al localStorage riceveremo un errore $QUOTA\_EXCEEDED\_ERR$.

I cookies, inoltre, sono parte del protocollo HTTP per cui i dati che contengono vengono inviati ad ogni richiesta, creando un notevole traffico.

I vantaggi principali di questo sistema, oltre alla semplicità di utilizzo, si vedranno soprattutto sui dispositivi mobili che potranno lavorare anche offline, ad esempio quando non c'è copertura di rete, su dati che poi potranno essere sincronizzati con la versione web.

2.8.3 Web Workers API

Quando il browser visualizza una pagina web, deve scaricarne tutti i componenti: immagini, file CSS, script Javascript, ecc. Il browser è in grado di scaricare in parallelo più di un file ma non gli script Javascript che vengono invece sempre scaricati uno per volta.

Questo tipo di markup ha un problema rilevante: se la risorsa esterna non risponde in tempi brevi, ad esempio perchè il server è sovraccarico o c'è un momentaneo problema di rete, il browser, chiamato

a visualizzare la pagina web che ospita tale codice, bloccherà il caricamento in attesa che il Javascript venga caricato. Ne consegue che tutto quello che si trova dopo uno script in una pagina web non può nè essere scaricato nè essere visualizzato fino al completo caricamento dello script stesso.

La soluzione a questo problema è offerta dalle Web Workers API che nascono per consentire l'esecuzione di porzioni di codice Javascript in modo asincrono, senza intaccare le performance della pagina web in visualizzazione. Grazie a questa tecnica il browser terminerà di caricare tutto il codice della pagina prima di iniziare il download del Javascript esterno. Così facendo, qualora la risorsa esterna che offre il codice risulti bloccata, la pagina sarà già caricata e l'unico elemento mancante sarà quello prodotto dal Javascript. Di conseguenza, il caricamento della pagina web diventa enormemente più veloce e non rischia di patire blocchi fastidiosi per l'utente.

I web workers sfruttano le moderne CPU multicore creando un nuovo thread nel quale sarà eseguito lo script. Per fare comunicare la pagina web vera e propria e i vari workers, HTML5 fornisce funzioni che permettono il passaggio dei dati.

Le API prevedono che un WebWorker possa essere generato come oggetto della classe Worker o SharedWorker: nel primo caso la sua esecuzione sarà limitata alla specifica sessione di navigazione all'interno della finestra del browser che l'ha invocato; nel secondo invece ogni sessione di navigazione che condivide la stessa origine (lo stesso dominio) potrà connettersi e scambiare messaggi con il medesimo worker.

In questo modo lo SharedWorker assume il ruolo di coordinatore, ottimo per, ad esempio, propagare su tutte le finestre del browser puntate su di un particolare dominio un messaggio ricevuto dal server. I metodi forniti per inviare e/o ricevere comunicazioni dai worker sono essenzialmente due:

1. postMessage()

2. onmessage();

Il primo è una funzione che se usata nella pagina web trasmette al worker, e se usata nel worker trasmette alla pagina web. L'argomento passato può essere di qualunque tipo, stringa, oggetto, numero, ecc. Il secondo è un evento che viene invocato quando si riceve un messaggio da postMessage() e può essere usato sia nel worker sia nella pagina web.

Oltre a queste funzioni nelle API sono presenti i comandi $onError()$ e $terminate()$. Entrambi questi comandi vanno utilizzati all'interno

della pagina web e permettono di gestire gli errori del worker e di terminarne l'esecuzione. I worker, infine, possono a loro volta eseguire workers. In questo caso gli eventi *onError*() e *terminate*() possono essere eseguiti nel worker 'padre'.

Dal punto di vista della sicurezza, questa è garantita dalla scarsa area di validità (o scope) di cui gode il componente: il web worker infatti non ha accesso alla gran parte dei componenti ed oggetti Javascript caricati nella pagina in cui gira e deve essere fornito tramite un file js esterno. Come si vede dalla figura i web workers non hanno la possibilità di accedere e manipolare gli oggetti window e document quindi non possono utilizzare le API per la gestione del DOM.

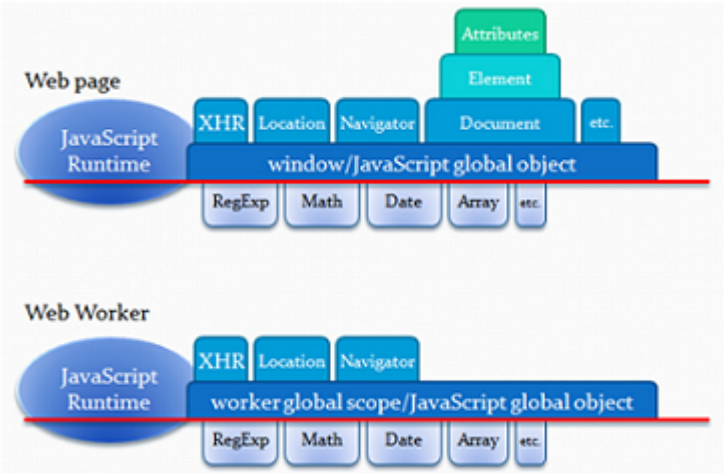


Fig. 9 - I diritti di accesso degli HTML 5 Web Workers

Non potendo accedere al DOM e non potendo modificare la pagina, viene da chiedersi a cosa serve spostare l'elaborazione su un Web Worker. La risposta è semplice: si fa fare l'elaborazione al Web Worker e si passano i risultati al thread principale che si occupa di aggiornare la pagina. In questo scenario, questa nuova funzionalità può essere di grande aiuto nei seguenti casi:

- elaborazione di immagini;
- gestione di una grande quantità di dati (immaginate i 100 mila punti di interesse da caricare su una mappa);
- analisi di testi (correzione ortografica su grandi testi, ricerca in in grossi dizionari);
- elaborazione di video.

Vediamo un esempio di cosa si può ottenere: la fontana è costituita da centinaia di punti che si muovono indipendentemente. Questo è sicuramente molto impegnativo a livello elaborativo.

Però, con l'aiuto dei web worker, funziona bene senza bloccare l'interfaccia del browser.

I web workers generano fotogrammi di animazione composti da particelle (ogni goccia d'acqua) e li inviano alla pagina web tramite postMessage() dove essi sono riuniti su un'unica tela. La pagina principale memorizza tutti i fotogrammi raccolti e li disegna a ogni intervallo.

2.8.4 WebSockets

Una delle caratteristiche principali del web è l'architettura client-server, una modalità di comunicazione dove c'è un client, ben definito, che fa richieste ad un server, il quale fornisce le risposte quando è interrogato. Questa struttura è perfetta per un web statico, fatto

di pagine collegate tra loro in cui la richiesta di nuovo contenuto è stimolata dal click su un link da parte dell'utente del browser.

Questa architettura non è adeguata per applicazioni complesse e ricche di comunicazioni bidirezionali o dove le informazioni potrebbero dover arrivare quando sono disponibili e non quando l'utente fa qualcosa (ad esempio in applicazioni finanziarie o in real-time).

HTML5 risolve in maniera efficiente questo problema offrendo agli sviluppatori una soluzione semplice, efficiente ed efficace: le WebSockets API che introducono, una funzionalità tra le più attese: la possibilità di stabilire e mantenere una connessione dati (asincrona) tra browser e server remoto sulla quale far transitare messaggi in entrambe le direzioni. Le API offrono un semplice meccanismo grazie all'oggetto WebSocket, al metodo send e all'evento onmessage.

La creazione di un nuovo WebSocket richiede come unico parametro obbligatorio l'url verso la quale si vuole stabilire la connessione. Il protocollo può essere ws o wss, dove il secondo indica la richiesta di una connessione sicura:

$$var\ webSocket\ =\ new\ WebSocket('ws://echo.websocket.org/');$$

La riga di codice appena scritta dice semplicemente di creare una connessione (socket) al server con cui abbiamo deciso di instaurare il nostro dialogo.

Una volta creato l'oggetto si hanno a disposizione due metodi per l'invio di informazioni e per la chiusura del canale di comunicazione:

```
webSocket.send("Hello world");
webSocket.close()
```

Per interagire invece con i messaggi ricevuti dal server e con il cambiamento di stato del WebSocket stesso possiamo utilizzare i listeners:

```
webSocket.onopen = function(event) {
 console.log("Connection opened!");
}

webSocket.onmessage = function(event) {
console.log("Server says " + event.data);
}

webSocket.onerror = function(event) {
console.log("Error!!");
}
```

```
webSocket.onclose = function(event) {
console.log("Connection closed!");
}
```

I WebSocket, sono nella loro estrema semplicità, degli strumenti incredibilmente potenti; a riprova di questo fatto la rete ha già incominciato a offrire interessanti prospettive di utilizzo nonostante l'innegabile giovinezza di queste API.

Tra le soluzioni degne di nota merita sicuramente una citazione *Pusher*, un servizio che offre la possibilità di gestire eventi realtime attraverso l'utilizzo di WebSocket. Altrettanto valido è $jsTerm$, un'applicazione che consente di collegarsi a server remoti utilizzando un proxy, scritto in node.js, tra il protocollo WebSocket e Telnet.

Recentemente lo sviluppo delle API è stato frenato dalla scoperta di una seria vulnerabilità legata al potenziale comportamento di un caching proxy che può essere indotto, attraverso l'utilizzo di WebSocket ad-hoc, a modificare il contenuto delle informazioni consegnate ad altri client.

La vulnerabilità è stata correttamente risolta dalla versione 0.7 del protocollo ma sfortunatamente le versioni 4 e 5 di Firefox, rilasciate prima del fix, hanno i websocket disabilitati per default; la versione 6 del browser dovrebbe però ripristinare il funzionamento $out - of - the - box$ di questa interessantissima feature.

2.8.5 Geolocation API

Le Geolocation API consentono la possibilità di ottenere dal browser le informazioni sulla posizione geografica dell'utente, non sono contenute all'interno dell'HTML5 ma fanno parte di quell'insieme di documenti che gravitano attorno alle specifiche.

Queste funzioni, in maniera molto semplice, consentono di ottenere latitudine, longitudine più una serie di altre informazioni per ogni utente che visita le pagine web.

Ci sono due metodi a disposizione che servono entrambi a ottenere la posizione corrente: getCurrentPosition, che fornisce il dato una sola volta, e watchPosition, che si attiva automaticamente quando cambia la posizione.

```
navigator.geolocation.getCurrentPosition
(gotPosition,errorGettingPosition,
{'enableHighAccuracy':true,'timeout':10000,'maximumAge':0});

navigator.geolocation.watchPosition
```

```
(gotPosition,errorGettingPosition,
{'enableHighAccuracy':true,'timeout':10000,'maximumAge':0});
```

Il primo argomento contiene i riferimenti a una funzione da eseguire nel caso di un corretto recupero delle informazioni; tale funzione accetta come argomento un oggetto di tipo Position che contiene tutte le informazioni recuperate.

```
function gotPosition(pos) {
   var outputStr =
     "latitude:"+ pos.coords.latitude +"\n"+
     "longitude:"+ pos.coords.longitude +"\n"+
     "accuracy:"+ pos.coords.accuracy +"\n"+
     "altitude:"+ pos.coords.altitude +"\n"+
     "altitudeAccuracy:"+ pos.coords.altitudeAccuracy +"\n"+
     "heading:"+ pos.coords.heading +"\n"+
     "speed:"+ pos.coords.speed +"";

     alert(outputStr);
    }
```

Il secondo argomento è opzionale, punta a una funzione che sarà invocata in caso di errore. La funzione preposta deve accettare come parametro un oggetto di tipo PositionError contenente un codice di errore.

```
function errorGettingPosition(err) {
   if(err.code == 1) {
     alert("L'utente non ha autorizzato");
   } else if(err.code == 2) {
     alert("Posizione non disponibile");
   } else if(err.code == 3) {
     alert("Timeout");
   } else {
     alert("ERRORE:" + err.message);
   }
}
```

L'ultimo parametro, può essere utilizzato per specificare alcune opzioni utilizzando una struttura opzione1: valore1, ... Le opzioni disponibili sono tre:

- enableHighAccuracy (true/false): questo flag può essere utilizzato per notificare allo user-agent la necessità o meno di ottenere dati il più accurati possibile.

  L'opzione è stata introdotta in quanto il recupero di informazioni più dettagliate richiede di solito maggiore dispendio di tempo ed energia e, in particolare su device mobile, potrebbe risultare fastidiosa per l'utente;

- timeout (millisecondi): l'opzione rappresenta il tempo massimo concesso al browser per recuperare la posizione dell'utente.

  In caso di fallimento sarà invocata la funzione associata al secondo argomento;

- maximuAge (millisecondi): indica al browser di effettuare una ricerca preventiva nella cache di un dato geospaziale non più vecchio dei millisecondi specificati. Se disponibile tale dato sarà restituito come posizione corrente, altrimenti sarà eseguita la procedura classica.

Lo standard prevede che il browser possa esporre le coordinate geografiche dell'utente fornitegli dal sistema operativo su cui è installato, qualora quest'ultimo consenta di determinare la posizione geografica mediante antenna GPS o WiFi.

Pertanto, affinchè un'applicazione web ottenga la posizione geografica, è necessario che il sistema operativo e il browser supportino tale funzionalità (le API applicative HTML5 richiedono l'uso di Javascript, per verificare la compatibilità del browser).

La richiesta della posizione geografica è sempre preceduta da una richiesta esplicita fatta all'utente pertanto, sulla questione privacy, è l'utente a dover accettare esplicitamente di voler fornire al browser questo dato.

Esistono svariati ambiti in cui la geolocation può trovare applicazione, ecco qualche esempio:

- 'Come raggiungerci' - la classica pagina di siti relativi ad esercizi commerciali, leggendo la posizione dell'utente gli si può presentare una mappa con il percorso per raggiungere in auto la vostra sede, cosa sicuramente più evoluta rispetto alla classica mappa.

- Geotagging - che consente di descrivere elementi culturali e soprattutto di identificare fisicamente nello spazio quanto pubblicato sul web.

- AroundMe - una famosa app su iPhone che offre una lista completa di tutti i servizi che si trovano in zona indicando la loro distanza dal punto in cui ci si trova.

Capitolo 3

Dart

3.1 Introduzione

Dart è il nuovo linguaggio di programmazione, proposto da Google, per creare applicazioni web strutturate. Sviluppato con obiettivi di semplicità, efficienza e scalabilità, si propone nel lungo periodo di rimpiazzare Javascript come linguaggio di scripting standard lato client.

La presentazione ufficiale è avvenuta il 10 ottobre 2011 alla GOTO Conference in Danimarca, l'evento che ormai da quindici anni riunisce professionisti del settore e che si propone di fornire una visione delle nuove tecnologie e dei nuovi trends nel mondo dello sviluppo del software non commerciale.

A presentarlo sono stati i due ingegneri che lo hanno creato, Gilad Bracha e Lars Bak, il primo è tra i creatori di Java, il secondo è l'ideatore di V8, la tecnologia che dà al browser Chrome la velocità che lo caratterizza.

Come spiega l'azienda, Dart nasce per aiutare gli sviluppatori a creare un linguaggio per la programmazione web strutturato ma flessibile, facendolo apparire subito familiare e naturale ai programmatori i quali potranno realizzare applicazioni capaci di offrire prestazioni elevate su qualsiasi browser moderno.

Il codice è stato progettato per essere simile a linguaggi già esistenti, come C# e Java, ma con la necessaria flessibilità dei linguaggi web-oriented; inoltre, guardando il sito ufficiale, possiamo renderci conto di quanto l'ambiente di sviluppo abbia in comune con il già consolidato linguaggio Javascript.

I progetti sviluppati potranno essere eseguiti con virtual machine DartVM che sarà presto integrata in Google Chrome, Dart potrà essere eseguito all'interno del codice HTML con l'apposito MIME type 'application/dart' da utilizzare nei tag $< script >$ per includere il

sorgente del linguaggio. Per il momento è possibile utilizzare questo linguaggio grazie a un compilatore (Dart Cross Compiler) che traduce il codice Dart in codice Javascript, il quale può essere eseguito all'interno del browser, in modo da permettere sia agli sviluppatori di prendere confidenza con l'ambiente, sia al linguaggio di diffondersi. Infine nel sito ufficiale è possibile utilizzare un'applicazione, chiamata Dartboard, che è in grado di eseguire codice Dart, in modo che si possano vedere, modificare ed eseguire programmi Dart di piccole dimensioni.

L'intero progetto è open-source e sul sito ufficiale è possibile scaricare gratuitamente un set di librerie di base e i tool preliminari per la compilazione e l'esecuzione di codice Dart. Il progetto è in fase iniziale e Google mira a raccogliere esperienze e feedback dagli sviluppatori per capirne le reali potenzialità.

3.2 Perchè Dart?

Col passare del tempo, il Web ha dovuto far fronte alla sempre più crescente richiesta di integrazione di funzionalità interattive e contenuti multimediali.

Per far ciò sono state sviluppate numerose tecnologie che di volta in volta hanno permesso l'integrazione delle funzionalità reputate maggiormente significative per l'evoluzione del Web. Il Web 2.0 è sostanzialmente l'unione di tutte queste tecnologie: HTML, Javascript, CSS, etc.

Ad esempio, per impostare uno stile di visualizzazione diverso a seconda del supporto sul quale viene visualizzata la pagina, dobbiamo utilizzare CSS:

```
<link rel='stylesheet' media='all'
   href='/static/css/base.min.css' />
<link rel='stylesheet' media='only screen and
   (max-width:800px)'href='/static/css/mobile.min.css'/>
if (window.matchMedia('only screen and
   (max-width:480px)').matches){
```

```
//Asynchronously provide experience optimized for phone
} else if (window.matchMedia('only screen and
   (min-width:481px)and'+'(max-width:1024px)').matches){
//Asynchronously provide experience optimized for table or
smaller screen
}else{
// Asynchronously provide full screen experience
}
```

per salvare dei dati sul file system, invece, dobbiamo usare Javascript:

```
window.requestFileSystem(TEMPORARY, 1048576, initFs, fsError);
function saveFile(arrayBuffer, filename, type, callback){
  fs.root.getFile(filename,{create:true},function(fileEntry){
    fileEntry.createWriter(function(fileWriter) {
      var bb = new BlobBuilder();
      bb.append(arrayBuffer);
      fileWriter.write(bb.getBlob(type));
    }, fsError);
    callback(fileEntry);
  }, fsError);
}
```

Il risultato finale, sebbene permetta la realizzazione di una vasta gamma di applicazioni e funzionalità e la gestione di dati anche complessi, è scarsamente e malamente strutturato.

In particolar modo, nel momento in cui si fanno interagire queste tecnologie le une con le altre, assistiamo ad una perdita della percezione modulare dell'applicazione, ad un aumento esponenziale della complessità dei controlli per il debugging, nonchè ad un appesantimento non trascurabile delle operazioni di caricamento.

Javascript, inoltre, permette lo sviluppo semplice e veloce di piccole applicazioni, ma è altamente sconsigliato nel caso in cui l'applicazione da realizzare sia mediamente complessa.

La mancanza di una tipizzazione rende, infatti, molto difficile la gestione del codice Javascript, rendendo di fatto impossibile la realizzazione di validi tool di programmazione o di un supporto per moduli, package o librerie. A favore di Javascript c'è però la capacità di essere eseguito su qualsiasi macchina senza la necessità di installazione di particolari programmi.

La necessità di un approccio strutturato dei linguaggi per il web, sia per gestire efficacemente la realizzazione di grandi applicazioni, sia per far fronte ad un Web sempre in rapida evoluzione, ha portato

all'ideazione di un linguaggio che incorporasse tutte le funzionalità fino ad ora gestite da tecnologie diverse ed eterogenee, che risultasse familiare ai programmatori e che fornisse quel supporto strutturale che permette una buona ingegnerizzazione delle applicazioni.

3.3 Obiettivi

Gli obiettivi principali alla base della creazione di Dart sono principalmente due:

- creare un linguaggio strutturato ma, allo stesso tempo, flessibile che possa essere familiare e di naturale utilizzo per gli sviluppatori abituali,

- fare in modo che possa offrire prestazioni elevate su qualsiasi sistema operativo e dispositivo, sia che si tratti di computer desktop o dispositivi portatili, fornendo strumenti che rendano Dart compatibile con i maggiori browser in circolazione.

Dart nasce anche con l'obiettivo di risolvere alcuni dei problemi che si presentano agli sviluppatori Web:

- spesso piccoli script evolvono senza controllo in applicazioni web di grandi dimensioni che non hanno una struttura ben ordinata e comprensibile. Queste applicazioni monolitiche non possono essere suddivise per far sì che diverse squadre di sviluppatori si concentrino sulla programmazione contemporanea di più parti dell'applicazione in modo indipendente. È difficile essere produttivi quando un'applicazione web diventa grande.

- il successo dei linguaggi di scripting è dovuto principalmente alla loro natura semplice che permette di scrivere codice velocemente; ciò comporta che generalmente lo scambio di informazioni tra differenti parti del sistema sia improvvisato e descritto in commenti all'interno del codice, piuttosto che essere definito nella struttura del linguaggio stesso, rendendo arduo se non impossibile a chiunque non sia l'autore del codice, capire e mantenere un particolare pezzo di programma.

- i linguaggi esistenti costringono lo sviluppatore a scegliere tra linguaggi statici o dinamici. I tradizionali linguaggi statici richiedono uno stile di programmazione rigoroso che di solito viene percepito troppo stringente ed eccessivamente vincolato.

- gli sviluppatori non sono mai stati in grado di creare sistemi omogenei che comprendano sia client che server, ad eccezione di pochi casi, come Node.js e Google Web Toolkit (GWT).
- linguaggi e format diversi comportano cambi di contesto che sono ingombranti e aggiungono complessità al processo di codifica

3.4 Caratteristiche principali

Le caratteristiche principali del linguaggio Dart includono: classi, tipi opzionali, librerie.

3.4.1 Classi

Classi e interfacce forniscono un meccanismo chiaro per definire in modo efficiente le API. Questi costrutti consentono l'incapsulamento e il riutilizzo di metodi e dati fornendo un insieme di blocchi riutilizzabili ed estensibili.

- un'interfaccia definisce un set di metodi e costanti, a volte ereditando da altre interfacce.
- una classe può implementare più interfacce, ma eredita solo da una singola superclasse.

L'esempio seguente definisce un'interfaccia, insieme a una classe e sottoclasse che lo implementano:

```
interface Shape {
  num perimeter();
}

class Rectangle implements Shape {
  final num height, width;
  // Compact constructor syntax.
  Rectangle(num this.height, num this.width);
  // Short function syntax.
  num perimeter() => 2*height + 2*width;
}

class Square extends Rectangle {
  Square(num size) : super(size, size);
}
```

3.4.2 Tipi opzionali

Una delle caratteristiche più innovative del linguaggio di programmazione Dart è l'uso di tipi opzionali. I programmatori Dart possono aggiungere opzionalmente al loro codice tipi statici.

Secondo le preferenze del programmatore e lo stadio dell'applicazione, il codice può migrare da un semplice non tipato e sperimentale prototipo a una complessa applicazione modulare con tipi.

Poichè i tipi stabiliscono l'intento del programmatore, serve meno documentazione per spiegare cosa succede nel codice e si possono usare strumenti di type-checking per il debugging.

Dart fornisce, a scelta del programmatore, un mix di verifiche statiche e dinamiche. Durante la sperimentazione, il programmatore può scrivere codice senza tipo per la prototipazione semplice. Via via che l'applicazione diventa più grande e più stabile, i tipi possono essere aggiunti per aiutare il debug e imporre una struttura dove desiderato.

Di seguito un esempio di codice non tipizzato in Dart, che crea una nuova classe Point che ha i parametri x e y e due metodi: scale() e distance().

```
class Point {
  var x, y;
  Point(this.x, this.y);
  scale(factor) => new Point(x*factor, y*factor);
  distance() => Math.sqrt(x*x + y*y);
}

main() {
  var a = new Point(2,3).scale(10);
  print(a.distance());
}
```

Questo è come il codice diventa con l'aggiunta dei tipi che garantiscono che x, y e factor siano di tipo num e che Point contenga due valori di tipo num.

```
class Point {
  num x, y;
  Point(num this.x, num this.y);
  Point scale(num factor) => new Point(x*factor, y*factor);
  num distance() => Math.sqrt(x*x + y*y);
}

void main() {
```

```
  Point a = new Point(2,3).scale(10);
  print(a.distance());
}
```

3.4.3 Librerie

Gli sviluppatori possono creare e utilizzare librerie che sono garantite non cambiare durante il runtime. Pezzi di codice sviluppato in modo indipendente possono quindi contare su librerie condivise.

Dart fornisce due set di librerie a supporto della programmazione:

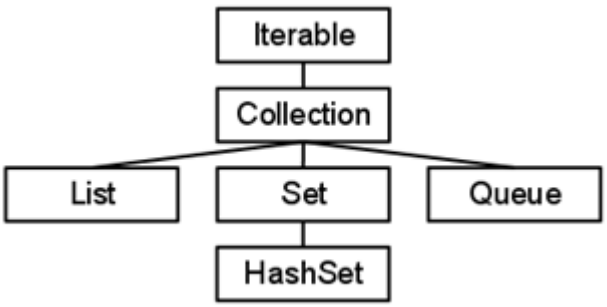
- *DOMLibrary* - contiene le interfacce per il DOM HTML5, basato liberamente sullo standard HTML5 specificato da

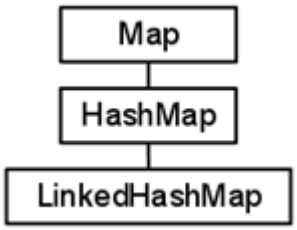
  W3C/WHATWG.

- *CoreLibrary* - contiene interfacce per supportare strutture dati comuni e operazioni.

La gerarchia dell'interfaccia della Core Library, attualmente ha tre parti principali che estendono le seguenti interfacce di base:

- Iterable - L'interfaccia Iterable permette di ottenere un Iterator da un oggetto Iterable; l'interfaccia viene utilizzato nei costrutti for-in per iterare su un oggetto Iterable; Collection è un interfaccia che implementa Iterable e definisce metodi implementati dagli interfaccia Set, List e Queue; HashSet è un'implementazione più specifica di Set



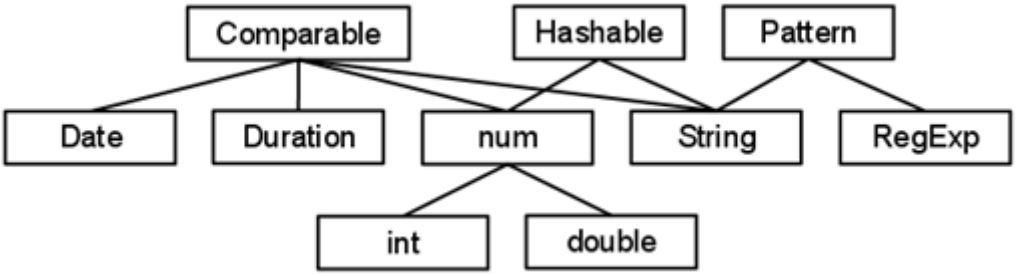


- Map - l'interfaccia Map è un contenitore associativo che associa una chiave ad un valore; l'interfaccia HashMap non garantisce un ordine particolare delle chiavi o dei valori restituiti dall'utilizzo di getKeys e getValues

- Comparable - l'interfaccia è implementato da Date, Duration, num e String

- Hashable - l'interfaccia è implementato da num e String

- Pattern - l'interfaccia è implementato da String e RegExp



3.4.4 Tooling

Dart includerà un set corposo di ambienti di esecuzione, librerie e strumenti di sviluppo costruiti per supportare il linguaggio. Questi strumenti permetteranno uno sviluppo produttivo e dinamico, includendo debug cambia-e-continua e oltre, fino a raggiungere uno stile dove tu programmi lo schizzo di un'applicazione, lo esegui e riempi i buchi mentre esegui.

3.5 Specifiche di linguaggo

Come già detto, Dart è un linguaggio a oggetti basato su classi con ereditarietà singola. La tipizzazione in Dart è opzionale e supporta interfacce e generici.

Il codice Dart può essere sottoposto a un'analisi statica alla ricerca di violazioni delle regole di tipizzazione, ma tali violazioni non precludono la compilazione o l'esecuzione del codice.

I programmi possono essere eseguiti in due modalità:

- di produzione (production mode), in cui le violazioni non hanno alcuna influenza sull'esecuzione;
- di controllo (checked mode), in cui le violazioni causano la generazione di eccezioni a runtime.

La coesistenza di queste due realtà è possibile per il semplice fatto che le regole di tipizzazione vengono utilizzate semplicemente come meccanismo di controllo e di individuazione rapida degli errori, ma possono essere completamente ignorate (a discrezione del programmatore) al fine dell'esecuzione del programma.

Scope lessicali

Lo scope lessicale di Dart è singolo, sia per le variabili, sia per le funzioni, sia per i tipi; non è permesso cioè avere due o più entità col medesimo nome. Dart permette di avere un'entità con lo stesso nome di una già esistente, solamente qualora si trovasse in uno scope più interno rispetto a quello contenente l'entità già esistente, tale situazione viene comunque segnalata tramite un warning statico. I nomi possono inoltre essere introdotti all'interno di un particolare scope attraverso meccanismi come gli import e l'ereditarietà.

.Privacy

Dart supporta la distinzione tra pubblico e privato. Una dichiarazione è privata se inizia con un underscore, altrimenti è pubblica; la privacy è indicata quindi tramite la nomenclatura di una data dichiarazione, permettendo sia agli umani che alle macchine di riconoscere i diritti di accesso alle dichiarazioni private al momento dell'utilizzo, senza curarsi del contesto dal quale esse sono derivate.

Concorrenza

Il codice di Dart è sempre a thread singolo. La concorrenza è supportata tramite entità chiamate isolates. Ogni isolato è dotato di una propria memoria e del proprio flusso di controllo; non c'è alcun tipo di stato condiviso e la comunicazione avviene tramite scambio di messaggi.

3.5.1 Errori e warning

Ci sono diverse tipologie di errori:

- errori di compilazione: sono errori che precludono la compilazione e devono essere segnalati dal compilatore Dart prima dell'esecuzione del codice. Le implementazioni di Dart lasciano molta libertà sulla scelta del momento di compilazione e, come avviene in molti linguaggi moderni, la compilazione di un metodo può essere rimandata alla sua prima invocazione, come conseguenza il rilevamento di questi errori potrebbe avvenire solo molto avanti nell'esecuzione. Come linguaggio web, Dart è spesso caricato direttamente da codice sorgente e per velocizzare il processo spesso si evita il parsing completo dei metodi. In un ambiente di sviluppo, però, tali errori dovrebbero essere indicati prontamente nell'interesse del programmatore.

  - warning statici: sono gli errori identificati durante il controllo statico e non hanno effetto sull'esecuzione. Molti di questi avvertimenti riguardano le regole di tipizzazione.

  - errori dinamici: sono gli errori identificati durante l'esecuzione in checked mode.

  - errori di esecuzione: sono errori che causano la generazione di eccezioni a runtime.

3.5.2 Variabili

In Dart le variabili possono essere precedute dai due modificatori static e final. Una variabile definita static non è associata a una particolare istanza ma alla classe o alla libreria. Una variabile definita final, invece, può essere definita solo una volta, alla sua inizializzazione. Una variabile può essere ovviamente sia static che final, l'utilizzo erroneo di questi modificatori genera un errore di compilazione. Nel caso in cui il tipo di una variabile non sia definito esplicitamente, la variabile è di tipo sconosciuto ed è associata al tipo Dynamic.

3.5.3 Funzioni

Tutte le funzioni hanno una signature e un corpo.La signature definisce i parametri formali della funzione il nome ed eventualmente anche il valore di ritorno; il corpo contiene invece le istruzioni eseguite dalla funzione.

Se l'ultima istruzione non è un'istruzione di ritorno, allora viene considerata implicita l'istruzione di ritorno del valore null.

Ogni dichiarazione di funzione include una lista di parametri formali, che consiste in una lista di parametri obbligatori e una di parametri opzionali. I parametri opzionali possono essere dotati di un valore di default, che però deve essere una costante o in fase di compilazione viene generato un errore; se non viene specificato nessun valore di default, allora viene implicitamente assegnato il valore di default null.

In casi di metodi astratti o di interfaccia, i parametri opzionali sono permessi, ma non è possibile assegnare loro un valore di default.

I nomi dei parametri opzionali non possono iniziare con un carattere: ciò permetterebbe la definizione di parametri opzionali con un nome che inizia per underscore, che risulterebbero quindi privati, rendendo impossibile l'estensione della funzione da parte di chiamanti esterni alla libreria.

Se non è esplicitamente assegnato un valore di ritorno alla funzione, tale valore di ritorno è definito Dynamic.

3.5.4 Classi

Una classe definisce la forma e il comportamento di un insieme di oggetti che sono le sue istanze; è dotata di costruttori, metodi, getter, setter e variabili. A parte i costruttori, gli altri membri possono essere sia membri di istanza che membri statici.

Ogni classe:

- ha una e una sola super-classe, fatta eccezione per la classe Object che non ha super-classe.
- può implementare una o più interfacce tramite la clausola implements.
- viene detta astratta se è definita esplicitamente tramite il modificatore abstract oppure se contiene almeno un metodo astratto (il modificatore abtract per le classi non è ancora implementato).
- non può dichiarare due membri con lo stesso nome, tranne nel caso in cui tali membri siano un getter e un setter entrambi

membri di istanza o entrambi membri statici. Ovviamente non è possibile definire metodi setter per una variabile final.

Dart permette la definizione di operatori personalizzati all'interno di una classe tramite il modificatore operator. Gli operatori non sono altro che metodi di istanza con nomi particolari, del tipo:

```
==, <, >, <=, >=, -, +, /, ~/, *, %, j, ^, &, <<,

>>, >>>, []=, [], ~, negate.
```

Gli operatori non hanno parametri opzionali.

I getter e i setter sono metodi particolari che servono a recuperare o definire il valore di una variabile e possono essere sia membri di istanza, riferendosi ad una variabile di istanza, sia membri statici, riferendosi ad una variabile statica. Un setter e un getter sono identificati rispettivamente dai modificatori set e get seguiti dall'identificatore della variabile cui si riferiscono, che dà loro anche il nome. Un getter o un setter non possono sovrascrivere un altro membro (in particolare un metodo) e viceversa.

I costruttori di una classe possono essere di tre tipi:

1. generativi: un costruttore generativo è un metodo denominato secondo il nome della classe cui appartiene e viene invocato tramite la parola chiave 'new', restituendo un'istanza della classe. Esempio di utilizzo di un costruttore generativo:

```
class A {
     var x;
     A([x]): this.x = x;
}
```

In alternativa è possibile utilizzare una forma contratta ed evitare di dover inizializzare ogni variabile:

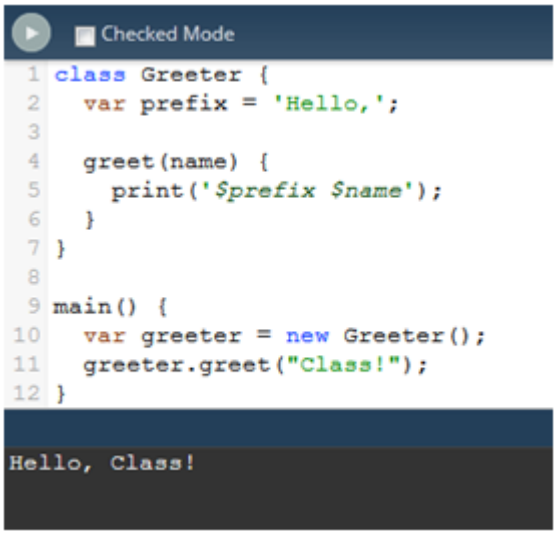
```
class A {
     var x;
     A([this.x]);
}
```

2. fabbriche: una fabbrica è un metodo statico, identificato dal modificatore factory, denominato secondo il nome della classe cui appartiene e, a differenza di un costruttore generativo, può restituire anche istanze di classi del sotto-tipo del valore di ritorno.

3. costanti: un costruttore costante, identificato dal modificatore const, viene utilizzato all'interno di classi di solo valori final per generare oggetti costanti in fase di compilazione. Esempio di utilizzo di un costruttore costante:

```
class C {
     nal x;
     nal y;
     nal z;
     const C(p, q): x = q, y = p + 100, z = p + q;
{
```

Vediamo un semplice esempio:



Il codice precedente mostra alcune caratteristiche di base delle classi Dart.

Nell'esempio è definita una classe, chiamata Greeter, la cui superclasse di default è Object.

Se c'è necessità di più di un costruttore per una classe, allora è possibile definire costruttori denominati ad esempio, Greeter.withPrefix().

Quando si definisce un costruttore, il costruttore senza argomenti di default non sarà creato, se serve, dovrà essere aggiunto:

```
// Nella classe Greeter:

Greeter();
Greeter.withPrefix(this.prefix);

// Nel codice:

var greeter = new Greeter.withPrefix('Howdy,');
```

In questo codice, $this.prefix$ è una scorciatoia che assegna il valore del parametro alla variabile prefix.

Proprio come per le variabili normali, creare variabili d'istanza, cioè dentro una classe, richiede l'utilizzo di var, final, o un tipo. Ogni oggetto Greeter ha la propria copia di un variabile chiamata $prefix$ che è inizializzata a $'Hello,'$.

È possibile impostare le variabili di istanza direttamente (ad esempio, greeter.prefix = 'Hi,'), o attraverso costruttori o con i metodi setter. *Setter* e *getter* sono un modo di fornire l'accesso ai dati senza esporre direttamente le variabili di istanza.

Ecco un esempio sul fornire un setter e getter per i dati prefisso nella classe Greeter:

```
class Greeter {
  String _prefix = 'Hello,';// var. d'istanza nascosta
  String get prefix() => _prefix; // Getter for prefix
  void set prefix(String value) { // Setter for prefix
    if (value == null) value = "";
    if (value.length > 20) throw 'Prefix too long!';
    _prefix = value;
  }

  greet(name) {
    print('$prefix $name');
  }
}

main() {
  var greeter = new Greeter();
  greeter.prefix = 'Howdy,';      // Setta prefix
  greeter.greet('setter!');
}
```

Si noti la speciale sintassi => per $prefix()$. Con questa notazione abbreviata il metodo restituisce il valore dell'espressione immediatamente dopo il =>.

Nel $main()$ viene creata una istanza invocando new seguito dal costruttore della classe Greeter. Poichè questo codice non definisce alcun costruttore Greeter, ottiene una versione predefinita che chiama il costruttore della superclasse senza argomenti.

La superclasse di Greeter è Object, il codice new Greeter () invoca il costruttore di Object (). Il metodo greet() definisce una funzione che è legata a un oggetto Greeter.

Specificare i tipi è facoltativo in Dart. I tipi possono aiutare a scrivere e gestire il codice, ma non cambiano il comportamento del programma.

3.5.5 Interfacce

Un'interfaccia definisce il modo con cui si interagisce con un oggetto; ha metodi, getter e setter, costruttori e opzionalmente anche un set di super-interfaccia. Non è possibile specificare alcun valore di default nella firma di un membro dell'interfaccia.

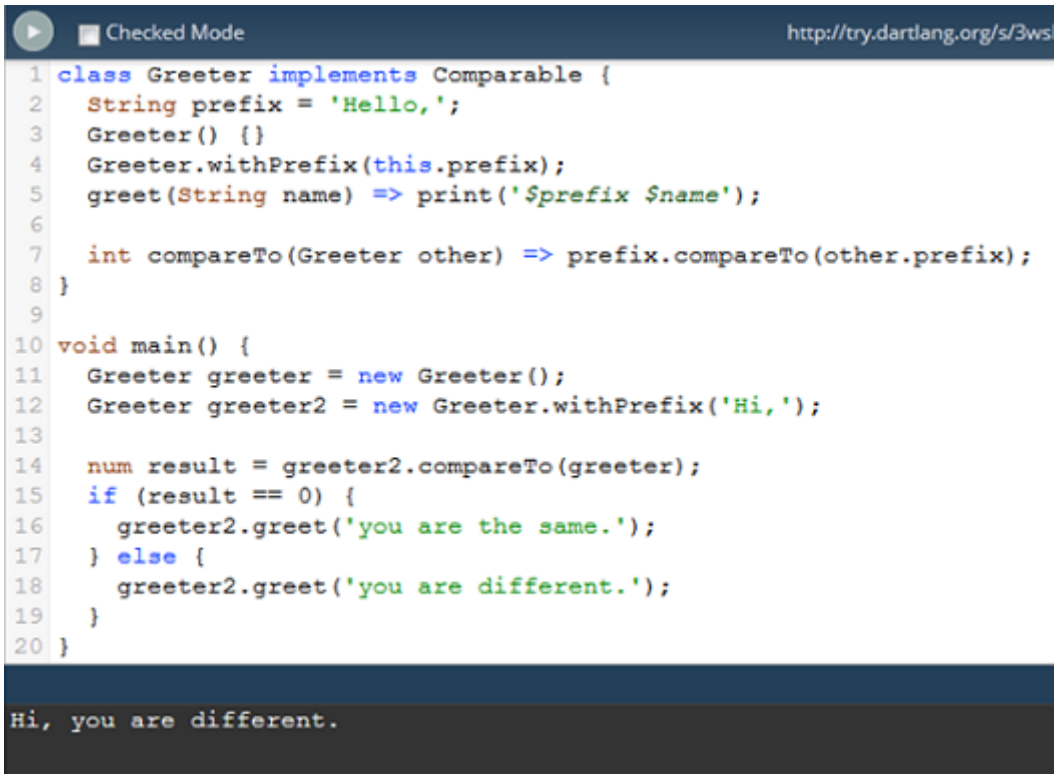
Al suo interno è possibile definire degli operatori e una classe fabbrica da utilizzare quando si cerchi di reificare una istanza richiamando il costruttore attraverso l'interfaccia. Per le interfacce valgono le stesse regole di controllo statico o a compile-time che valgono per le classi.

In Dart spesso è possibile creare oggetti direttamente da un'interfaccia, invece di dover trovare una classe che implementa tale interfaccia.

Questo è possibile perchè molte interfacce hanno una factory class, una classe che crea oggetti che implementano l'interfaccia. Per esempio, se il codice dice $newDate.now()$, la classe fabbrica per l'interfaccia di Data crea un oggetto che rappresenta l'ora corrente.

Il codice seguente usa e implementa interfacce. La classe Greeter implementa l'interfaccia Comparable. L'implementazione avviene in due fasi: aggiungendo *implementsComparable* alla dichiarazione di classe (linea 1), e aggiungendo una definizione del solo metodo richiesto da Comparable: compareTo () (linea 7).

int e double non sono tipi primitivi, sono in realtà interfacce che estendono l'interfaccia num. Ciò significa che le variabili int e double sono anche nums. è importante assicurarsi di inizializzare i numeri perchè sono oggetti per cui il loro valore iniziale è null, non 0.



3.5.6 Generici

Dart supporta i generici. Una classe, un'interfaccia o un membro d'istanza possono essere generici, utilizzare cioè parametri formali che sottintendono una famiglia di dichiarazioni, una per ogni set di parametri veri e propri forniti nel programma.

Un parametro può essere seguito dalla clausola extends, per indicare un upper bound del parametro; se la clausola extends non è presente, è sottinteso che la super-classe sia Object.

Nel caso in cui si cerchi di definire un upper bound per un parametro che è un super-tipo di quello stesso upper bound, si genera un warning statico.

3.5.7 Espressioni

Un'espressione è un frammento di codice che può essere valutato durante l'esecuzione per ottenere un valore, che è sempre un oggetto.

Costanti

Una costante è un'espressione il cui valore non può mai cambiare e che può essere valutato interamente in fase di compilazione. Una costante

potrà essere:

- un numero
- un booleano
- una stringa (escludendo i casi di stringhe soggette a interpolazione)
- il valore null
- un riferimento ad una variabile final e static
- l'invocazione di un costruttore costante
- una mappa o una lista costanti
- operatori con termini costanti

Null

L'oggetto null è la sola istanza della classe Null. Non è possibile istanziare, estendere, implementare Null, nè invocare quindi alcun metodo su un oggetto null a meno che tale metodo non sia già implementato nella classe.

A qualsiasi oggetto può essere assegnato il valore null.

Ecco un esempio di come si potrebbe implementare la classe:

```
class Null {
   factory Null._() throw "cannot be instantiated";
   noSuchMethod(InvocationMirror msg){
      throw new NullPointerException();
   }
/* other methods, such as == */
}
```

Numeri

Un numero è un intero decimale o esadecimale (se preceduto da 0x o 0X) di grandezza arbitraria o un decimale a 64 bit.

Per ragioni di chiarezza e di compatibilità con Javascript, i numeri possono essere preceduti da un segno +, ma ciò non ha alcun significato semantico. In Dart gli interi non hanno una grandezza massima prefissata e sono limitati solo dalla memoria a disposizione.

Non è possibile estendere o implementare int o double e solamente int e double possono estendere num.

Booleani

Le parole chiave true e false denotano i due rispettivi valori booleani. Sono le uniche implementazioni permesse dell'interfaccia bool.

In Dart qualsiasi oggetto può essere ricondotto a un valore booleano secondo quella che viene chiamata conversione booleana.

La conversione booleana è necessaria trovandosi in un ambiente a tipizzazione dinamica per evitare ambiguità come, per esempio, in Javascript, dove, a causa della tecnica chiamata auto-boxing, qualsiasi oggetto non null o numero non zero, sono valutati come true; in alcuni scenari particolari è possibile addirittura far valutare il valore false come true.

Attraverso la conversione booleana, un oggetto è ricondotto a true solo se corrisponde effettivamente a tale valore.

Stringhe

Una stringa è una sequenza di caratteri Unicode delimitati da apici singoli o doppi e può essere a riga singola o multipla.

In una stringa è possibile inserire caratteri di escape preceduti dal backslash, oppure è possibile premettere alla stringa il carattere @ identificandola come stringa raw, cioè come stringa in cui il carattere backslash non identifica caratteri di escape.

Tutte le stringhe implementano l'interfaccia String e per l'utente non è possibile implementare o estendere tale interfaccia.

Lista dei meta-caratteri:

```
\n newline, equivalente a \x0A.
\r ritorno a capo, equivalente a \x0D.
\f for nuova pagina, equivalente a \x0C.
\b for backspace, equivalente a \x08.
\t for tab, equivalente a \x09.
\v for vertical tab, equivalente a \x0B
\x numero esadecimale a 2 cifre
\u numero esadecimale a 4 cifre
\u{NUMERO_ESADECIMALE} il numero in esadecimale
rappresentato da NUMERO_ESADECIMALE
$ inizio di una interpolazione
in tutti lgi altri casi, \k indica il carattere
k per ogni k non in {n; r; f; b; t; v; x; u}.
```

In Dart esiste un meccanismo chiamato interpolazione (string interpolation) che permette di includere un'espressione all'interno di una stringa.

L'interpolazione viene identificata attraverso l'utilizzo del carattere $ all'interno di una stringa, seguito da una espressione racchiusa tra parentesi graffe.

L'interpolazione dovrebbe essere preferita all'utilizzo del comune operatore + poichè, in un contesto a tipizzazione dinamica, l'utilizzo dell'operatore + richiede un'analisi dinamica del codice per identificare la necessità di un'operazione algebrica o di concatenazione tra stringhe, mentre attraverso l'utilizzo dell'interpolazione, tale analisi può essere effettuata staticamente aumentando notevolmente la velocità di esecuzione, specialmente per un linguaggio che deve essere compilato in Javascript.

Liste

Una lista è un elenco indicizzato di oggetti. Una lista può avere 0 o più elementi e può essere preceduta dalla parola chiave const, in quel caso è una lista costante e deve contenere solamente elementi costanti per permettere la valutazione in fase di compilazione.

Se due liste contengono gli stessi elementi, allora sono la stessa lista, cioè in Dart le liste sono canonicizzate.

Sia perchè spesso la compilazione avviene in realtà in fase di esecuzione sia per agevolare l'utilizzo di liste a contenuto mutevole, si è deciso che tutte le liste siano considerate parametrizzate con tipo Dynamic.

Nel caso in cui si voglia annidare una lista dentro un'altra, è possibile specificare la parametrizzazione per la lista più esterna, ma le liste annidate all'interno saranno considerate ugualmente Dynamic.

Mappe

Una mappa è una struttura che associa stringhe a oggetti. In una mappa abbiamo 0 o più elementi, costituiti da una chiave (una stringa) e un valore (un oggetto).

Una mappa può essere preceduta dalla parola chiave const ed essere una mappa costante, contenente cioè solo valori costanti valutabili a tempo di compilazione. Nel caso di valori duplicati delle chiavi, l'ordine è definito dalla prima occorrenza, mentre il valore è definito dall'ultima.

Anche per le mappe valgono le stesse considerazione fatte per le liste per quanto concerne l'utilizzo del tipo Dynamic.

Funzioni

Una funzione, intesa come ciò che implementa l'interfaccia Function, è un oggetto che incapsula una unità di codice Dart. Se non diversamente specificato, anche in questo caso il tipo del valore di ritorno viene considerato Dynamic.

Creazione di istanze

Le espressioni di creazione di istanza invocano i costruttori per produrre delle istanze. Non è possibile ovviamente istanziare classi astratte e ci sono fondamentalmente due modi per istanziare una classe: utilizzando new o utilizzando const.

L'utilizzo di new ha l'effetto di invocare il costruttore della classe che si sta cercando di inizializzare e di allocarne una nuova istanza. L'utilizzo di const, invece, produce una istanza della classe costante a cui il costruttore si riferisce; le classi costanti sono canonicizzate e, se al momento della creazione dell'istanza Dart si accorge che è già presenta una istanza di tale classe, allora l'istanza appena creata viene abbandonata in favore di quella creata in precedenza.

Creazione di un isolato

La creazione di un isolato avviene attraverso quella che è una semplice chiamata di una libreria, invocando il metodo spawn() definito nella classe Isolate. Ogni isolato è dotato di memoria separata dagli altri isolati e di un personale thread di controllo.

Estrazione di proprietà

L'estrazione di proprietà permette ad un membro di un oggetto di essere estratto concisamente dall'oggetto stesso. L'estrazione può avvenire o attraverso l'utilizzo di un metodo oppure di un getter membri ach'essi dell'oggetto. La scelta di implementare una proprietà attraverso un metodo o un getter è riflettuto nell'interfaccia della classe.

Invocazione di funzioni

L'invocazione di una funzione può avvenire in uno dei seguenti tre casi:

- quando viene invocata una espressione funzione,
- quando viene invocato un metodo,
- quando viene invocato un costruttore.

La differenza sta nel modo in cui la funzione viene invocata e se *this* sia identificato in una istanza particolare o meno. Una volta identificata la funzione, i parametri sono assegnati al loro effettivo valore e comincia l'esecuzione del corpo della funzione secondo gli assegnamenti.

L'esecuzione termina quando si verifica una delle seguenti possibilità:

- viene lanciata una eccezione non gestita
- viene raggiunta una istruzione di return annidata nel corpo della funzione
- viene raggiunta l'ultima istruzione del corpo della funzione

Invocazione di metodi

L'invocazione di un metodo può prendere diverse forme.

- *Invocazione standard*: un'invocazione di metodo standard ha l'effetto di ricercare il metodo all'interno della classe di cui è istanza l'oggetto sul quale avviene l'invocazione.; se la ricerca fallisce, allora si procede a ricercare il metodo nella super-classe, procedendo in questo modo fino alla prima occorrenza del metodo cercato. Se la ricerca non ha buon fine, allora è sollevata un'eccezione.
- *Invocazione statica*: l 'invocazione statica di un metodo avviene direttamente sulla classe e non sull'oggetto istanza. A differenza dell'invocazione standard, la ricerca del metodo avviene solo nella classe in questione, non coinvolgendo le eventuali superclassi: in altre parole poichè l'invocazione abbia successo, il metodo deve essere definito direttamente nella classe sulla quale avviene l'invocazione. Anche in questo caso, nell'eventualità che l'invocazione fallisca, è sollevata un'eccezione.
- *Invocazione tramite super*: l'invocazione di un metodo tramite super è analoga all'invocazione standard, ma la ricerca del metodo invocato inizia dalla classe a cui fa riferimento super, anzichè quella a cui fa riferimento this. Anche in questo caso, nell'eventualità che l'invocazione fallisca, è sollevata un'eccezione.
- *Invio di messaggi*: i messaggi sono l'unico metodo di comunicazione tra isolati.Essi sono inviati attraverso l'invocazione di metodi specifici nelle librerie Dart.

I metodi che supportano l'invio di messaggi utilizzano delle primitive Dart che non sono accessibili da codice normale, come nel caso dei metodi per la creazione di isolati.

Invocazione di un getter

Un getter fornisce accesso al valore di una proprietà. L'invocazione di un getter procede esattamente come l'invocazione standard di un metodo: il getter è ricercato all'interno della classe di cui l'oggetto su cui invochiamo il getter è istanza, risalendo le super-classi fino alla prima occorrenza del getter. Nell'eventualità che l'invocazione fallisca, è sollevata una eccezione.

Assegnamento

Un assegnamento cambia il valore associato a una variabile o a una proprietà ed è identificato tramite il simbolo =. Dart supporta gli assegnamenti compositi, gli assegnamenti, cioè, in cui il segno = è preceduto da un operatore. Lista dei più comuni operatori di assegnamento composti:

```
‘*=’ moltiplicazione
‘/=’ divisione
‘%=’ resto
‘+=’ addizione
‘-=’ sottrazione
‘<<=’ shift a sinistra
‘>>=’ shift a destra
‘&=’ and bit a bit
‘^=’ negazione bit a bit
’|=’ or bit a bit
```

Espressione condizionale

L'espressione condizionale è una particolare espressione del tipo

$'expr1?expr2 : expr3'$ che identifica una forma altamente concisa del costrutto IF-THEN-ELSE: se l'espressione che precede il segno $'?'$ viene valutata a true, allora viene eseguita l'espressione a sinistra del segno $' :'$, altrimenti viene eseguita quella a destra. Nel caso in cui l'oggetto in cui è valutata l'espressione a sinistra di $'?'$ non sia un booleano, allora l'oggetto è sottoposto a conversione booleana.

Espressioni logiche booleane

Un'espressione logica booleana combina oggetti booleani utilizzando gli operatori di congiunzione $'\&\&'$ o disgiunzione $'||'$ e può essere un'espressione di tipo bitwise o un'invocazione di un operatore booleano su di un'espressione generica.

Espressioni bitwise

Le espressioni bitwise invocano gli operatori bitwise sugli oggetti. Un'espressione bitwise può essere o un'espressione di uguaglianza o l'invocazione di un operatore bitwise su super o su di un'espressione generica.

Espressioni di uguaglianza

Le espressioni di uguaglianza testano gli oggetti in cerca di identità o di uguaglianza e possono essere o una espressione relazionale oppure l'invocazione di un operatore di uguaglianza su super o su di una espressione generica.

Nel caso in cui le espressioni argomento non si valutino in un valore booleano allora viene utilizzata la conversione booleana. '==', '!=' operatori di identità: controllano che due variabili contengano, o meno, lo stesso oggetto '===', '!==' operatori di uguaglianza: controllano che due oggetti abbiano, o meno, lo stesso valore

Espressioni relazionali

Un'espressione relazionale invoca un operatore relazionale su un oggetto. Un'espressione relazionale può essere un'espressione shift oppure l'invocazione di un operatore relazionale su super o su di una espressione generica.

Espressione shift

Una espressione shift invoca un operatore di shift su un oggetto. Una espressione shift può essere una espressione additiva oppure l'invocazione di un operatore di shift su super o su di una espressione generica.

Espressioni additive

Una espressione additiva invoca un operatore di addizione su un oggetto. Una espressione additiva è una espressione moltiplicativa op-

pure l'invocazione di un operatore di addizione su super o su di una espressione generica.

Espressioni moltiplicative

Le espressioni moltiplicative invocano gli operatori di moltiplicazione sugli oggetti. Una espressione moltiplicativa è una espressione unaria oppure l'invocazione di un operatore di moltiplicazione su super o su di una espressione generica.

Espressioni unarie

Le espressioni unarie invocano operatori unari sugli oggetti. Una espressione unaria può essere una espressione prefissa o post-fissa, l'invocazione di un operatore di incremento o l'invocazione di una espressione o di un operatore unario su super o su di una espressione generica.

Espressioni post-fisse

Le espressioni post-fisse invocano gli operatori post-fissi sugli oggetti. Una espressione post-fissa può essere una espressione primaria, una funzione, l'invocazione di un metodo o di un getter o l'invocazione di un operatore post-fisso su di una espressione generica.

Espressioni assegnabili

Le espressioni assegnabili sono espressioni che possono comparire sul lato sinistro di un assegnamento. Tipicamente le espressioni assegnabili sono sotto-espressioni il cui valore deve essere valutato. Una espressione assegnabile può essere:

- un identificatore
- l'invocazione di un metodo, di un getter o un operatore di accesso a lista su di una espressione generica
- l'invocazione di un getter o di un operatore di accesso a lista su super

Identificatori

Un identificatore permette l'accesso ad un oggetto attraverso un nome non qualificato. Dart possiede degli identificatori built-in, cioè delle parole chiave di Dart, che però non risultano parole riservate in Javascript.

Per minimizzare le incompatibilità dovute al porting di codice Javascript verso Dart, gli identificatori built-in non sono parole riservate nemmeno in Dart: è possibile utilizzarle tranquillamente tranne che per dare il nome ad un tipo o ad una classe.

Test dei tipi

L'espressione is viene utilizzata per verificare che un oggetto appartenga ad un determinato tipo. Applicare l'espressione al tipo Object restituirà sempre un valore true.

3.5.8 Dichiarazioni

Dart supporta tutti i tipi di dichiarazioni standard dei più comuni linguaggi di programmazione:

- blocchi - sequenze di istruzioni racchiuse da parentesi graffe
- espressioni - singole istruzioni che racchiudono una espressione valutabile in un oggetto
- dichiarazioni di variabile - istruzioni che introducono una nuova variabile nello scope lessicale corrente
- if - dichiarazione che permette l'esecuzione condizionale di sotto-blocchi di istruzioni
- for - esecuzione iterativa di un blocco di istruzioni utilizzando una variabile di iterazione
- foreach - esecuzione iterativa di un blocco di istruzioni utilizzando un iteratore su una lista
- while - esecuzione iterativa di un blocco di istruzioni dove la condizione di esecuzione è verificata prima del ciclo
- do - esecuzione iterativa di un blocco di istruzioni dove la condizione di esecuzioneè verificata dopo il ciclo
- switch - esecuzione condizionale di sotto-blocchi di istruzioni gestendo un grande numero di alternative
- etichette - possibilit di assegnare una etichetta alle alternative di uno switch
- break - istruzione che forza l'uscita dalla struttura ciclica in cui ci si trova

- continue - istruzione che forza la terminazione dell'iterazione corrente all'interno della struttura ciclica in cui ci si trova
- try - esecuzione di un blocco di istruzioni prevedendo la possibilit di generazione di eccezioni
- catch - possibilità di catturare le eccezioni generate all'interno di una dichiarazione try
- throw - possibilità per l'utente di generare eccezioni autonomamente
- return - istruzione di ritorno della chiamata di una funzione
- assert - possibilità (in modalità di controllo) di interrompere il normale flusso di esecuzione attraverso la verifica di condizioni predefinite dall'utente

3.5.9 Librerie e script

Una libreria è costituita da un set di import e un set di dichiarazioni di primo livello. Una dichiarazione di primo livello può essere una classe, un'interfaccia, una dichiarazione di tipo, una funzione o una dichiarazione di variabile. Una libreria può contenere uno script tag, identificato dai caratteri '!', che specifica all'ambiente nel quale si trova lo script quale interprete utilizzare.

Una libreria è un'unità di privacy (intesa secondo Dart); è impossibile cioè accedere a dei membri privati della libreria dall'esterno. Lo scope di una libreria è costituito dalle dichiarazioni di primo livello e dai nomi introdotti dagli import.

Import

Un import specifica una libreria da utilizzare all'interno di un'altra libreria. L'import specifica un URI al quale trovare la libreria e presuppone uno spazio dei nomi delle librerie globale (almeno all'interno dell'isolato corrente).

Lo spazio dei nomi introdotti attraverso un import potrebbe entrare in conflitto con le dichiarazioni locali, o con gli spazi dei nomi introdotti da altri import, per cui è vietato introdurre membri con nomi gi utilizzati nelle librerie importate o importare librerie che utilizzano gli stessi nomi.

Include

Una direttiva include specifica un URI al quale è possibile trovare una unità di compilazione Dart da incorporare nella corrente libreria. Una unità di compilazione è una serie di dichiarazioni di primo livello. La compilazione di una direttiva include causa la compilazione di ciò che si trova all'URI specificato e l'inserimento all'interno dello scope corrente.

Script

Uno script non è altro che una libreria con una funzione main() di primo livello. Uno script viene compilato come una libreria qualsiasi, dopodichè viene invocata la funzione main(). I nomi per gli script sono opzionali.

3.5.10 Tipi

Dart supporta una tipizzazione opzionale basata sui tipi dell'interfaccia. Il sistema di tipizzazione non è completo a causa della presenza dei generici e, in questi casi, il controllo completo sulla tipizzazione è delegato ai tool di programmazione utilizzati.

Tipi statici

Le annotazioni di tipo statico sono utilizzate nella dichiarazione di variabili e nelle istruzioni di ritorno delle funzioni. Le annotazioni di tipo statico sono utilizzate solamente in modalità di controllo e non hanno alcuna influenza in modalità di produzione. Un'implementazione di Dart deve fornire un controllore statico in grado di individuare quei casi che sono identificati come avvertimenti statici, ma:

1. eseguire un controllo statico su di un programma non è assolutamente necessario per la compilazione e l'esecuzione di tale programma

2. anche nel caso in cui il controllo statico identificasse dei problemi, questo non deve precludere la corretta compilazione ed esecuzione del programma

Tipi dinamici

Un'implementazione di Dart deve supportare l'esecuzione sia in modalità di controllo che in modalit di produzione. I controlli che sono

specifici della modalità di controllo devono avvenire solamente se il codice viene eseguito in modalità di controllo. Un tipo T è malformato se e solo se:

- T è della forma id e id non denota alcun tipo all'interno dello scope lessicale corrente
- T è un tipo parametrizzato della forma G $< S1, ..., Sn >$ e:
    1. G o uno degli Si sono malformati
    2. G non è un tipo generico con n parametri
    3. Si non è un sotto-tipo dei parametri di tipo di G

In modalit di controllo viene generato un errore se si cerca di utilizzare un tipo malformato. In modalit di produzione, tuttavia, qualsiasi tipo malformato viene trattato come una istanza di Dynamic, permettendo ugualmente l'esecuzione del codice.

Alias di tipo

Dart permette la definizione di alias di tipo associati a funzioni attraverso l'utilizzo della parola chiave typedef. L'effetto della dichiarazione di un alias è quello di inserire il tipo definito nello scope lessicale associandolo al tipo del valore di ritorno della funzione. Se la funzione non definisce un tipo del valore di ritorno, allora l'alias è associato al tipo Dynamic.

Dynamic

Il tipo Dynamic identifica il tipo sconosciuto. Se non viene fornita alcuna dichiarazione statica di tipo o se si utilizza un tipo generico senza fornire i corrispondenti tipi degli argomenti, allora il sistema di tipizzazione identifica un caso di tipo sconosciuto. Il tipo Dynamic ha metodi per qualsiasi possibile identificatore e arità, con ogni possibile combinazione di parametri nominali. Questi metodi hanno tutti Dynamic come tipo di ritorno e i loro parametri formali sono tutti di tipi Dynamic. Il tipo Dynamic ha proprietà per ogni possibile identificatore e sono tutte di tipo Dynamic.

Void

Il tipo void viene utilizzato solamente come valore di ritorno delle funzioni. Non è possibile accedere a membri del risultato dell'invocazione di un metodo void, quindi assegnare il risultato di un metodo void ad una variabile o passarlo come parametro generer un avvertimento

statico, a parte il caso in cui la variabile o il parametro formale siano di tipo Dynamic; è possibile però restituire il risultato di un metodo void dall'interno di un metodo void o restituire null o un valore di tipo Dynamic

3.6 Server side

Dart mette a disposizione una libreria finalizzata a supportare la computazione lato server. È possibile infatti eseguire una VM in grado di eseguire servizi anch'essi scritti in Dart. Nella libreria sono incluse API per la gestione del file system, di stream di dati, socket, processi e thread. La VM è in grado di gestire le più comuni funzionalità di un'applicazione distribuita di tipo client-server come ad esempio l'elaborazione di richiesta HTTP o il recupero di dati inviati tramite socket stream.

3.7 Applicazioni di Dart

Una delle principali ragioni che hanno portato alla decisione di sviluppare Dart è stata quella di fornire un linguaggio in grado di risolvere le problematiche che erano emerse dall'evoluzione del Web, tra cui sopperire alla mancanza di un linguaggio di programmazione/scripting valido che potesse essere utilizzato sia lato client che lato server. Di seguito verrà illustrato il modo in cui Dart si collocherà nel paradigma del Web, le sue principali differenze e analogie con Javascript, i miglioramenti introdotti dal linguaggio e le opportunità di utilizzo in ambito client-server.

3.7.1 Incorporare Dart in HTML5

I tag script di HTML forniscono un attributo type per definire il linguaggio dello script. Per Dart, questo attributo ha il valore 'application/dart'. Come per altri tag per script, il contenuto può essere inserito come corpo del tag script o specificato con un URL utilizzando l'attributo src. Lo script Dart dovrà avere una funzione di primo livello *main*() o dichiarata direttamente nello script o in un file importato. Il browser invoca il *main*() al caricamento.

Lo script Dart potrà avere ulteriori comandi *source* e *import* per includere altri script o librerie. Incorporare il codice Dart è differente da incorporare codice Javascript. Ogni pagina HTML può avere più tag script Dart, ma ogni tag script nella pagina viene eseguito

isolatamente. In Javascript, le dichiarazioni in ogni tag vengono combinate nello stesso namespace. In Dart il codice all'interno di un tag script non può accedere al codice in un altro tag tranne che utilizzando *source* o *import* ed ogni script deve avere il proprio *main*() perchè possa essere eseguito. Ogni *main*() viene invocato in concomitanza dell'evento DOMContentLoaded , questo si riflette in un ordinamento non predefinito dell'esecuzioni dei vari script, ma questo permette il caricamente asincrono/concorrente. Il codice Dart viene eseguito solo dopo che la pagina è stata elaborata, i programmatori Dart, quindi, possono assumere che il DOM sia caricato completamente. Con Javascript, i programmatori possono inserire ascoltatori di eventi direttamente nell' HTML, ma questo è generalmente scoraggiato. Anche con Dart queto è permesso ma scoraggiato, poichè la pagina HTML viene caricata molto più velocemente se gli ascoltatori sono aggiunti in seguito separatamente.

```
<html>
  <body>
    <script type='application/dart'>
      void main() {
        HTMLElement element = document.getElementById('message');
        element.innerHTML = 'Hello from Dart';
      }
     </script>
     <div id='message'></div>
  </body>
</html>
```

In questo esempio viene presentato uno script che definisce un proprio isolato/libreria. La modalità di invocazione del main() garantisce che il div richiamato nello script esista al momento dell'esecuzione.

```
<html>
  <body>
    <script type='application/dart'>
      void hello(String text) {
        HTMLElement element = document.getElementById('message');
        element.innerHTML = text;
      }//This tag triggers a warning as no main() is defined.
    </script>
    <script type="application/dart">
      void main() {
        hello('Hello from Dart');//hello() will not resolve.
      }
    </script>
    <div id="message"></div>
  </body>
</html>
```

In questo esempio invece il main() viene definito in un tag script diverso; genererà quindi un errore, non godendo Dart del namespace condiviso come Javascript.

3.7.2 Miglioramento del DOM

Uno dei cambiamenti più semplici è stato quello di aver eliminato alcuni nomi scomodi. HTMLElement è diventato Element ed è stato eliminato HTML dalla maggior parte dei nomi dei tipi. Invece che childNodes e children ora abbiamo nodes e elements, mentre ownerDocument è solamente document. I nomi sono stati ottimizzati così che le cose più frequentemente utilizzate siano più facili. Fino ad ora per trovare elementi all'interno del DOM si dovevano ricordare una serie infinita di metodi:

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

Con Dart tutti questi metodi si sono ridotti a due: query() e queryAll().

```
// Old:
elem.getElementById('foo');
elem.getElementsByTagName('div');
elem.getElementsByName('foo');
elem.getElementsByClassName('foo');
elem.querySelector('.foo .bar');
elem.querySelectorAll('.foo .bar');
// New:
elem.query('#foo');
elem.queryAll('div');
elem.queryAll('[name="foo"]');
elem.queryAll('.foo');
elem.query('.foo .bar');
elem.queryAll('.foo .bar');
```

In Javascript, i tipi delle collezioni del DOM sono differenti dal tipo Array built-in, il chè fa impazzire i programmatori. In Dart, metodi come elements, nodes e query() che ritornano collezioni, ritornano oggetti che implementno l'interfaccia built-in di Dart per le collezioni, cioè liste, mappe e set Dart. Questo ha permesso di eliminare molti metodi specifici:

```
// Old:
elem.hasAttribute('name');
elem.getAttribute('name')
elem.setAttribute('name', 'value');
elem.removeAttribute('name')
// New:
elem.attributes.contains('name');
elem.attributes['name'];
elem.attributes['name'] = 'value';
elem.attributes.remove('name');
```

Analogamente:

```
// Old:
elem.hasChildNodes();
elem.firstChild();
elem.appendChild(child);
// New:
elem.nodes.isEmpty();
elem.nodes[0];
elem.nodes.add(child);
```

Creare elementi HTML è molto più semplice, grazie all'utilizzo di classi factory:

```
// Old:
document.createElement('div');
// New:
new Element.tag('div');
```

Oppure è possibile fare anche:

```
TableElement table = new Element.html('<table>
                  <tr><td>Hello <em>Dart!</em></table>');
```

Il cambiamento più importante e utile è quello di aver ripulito il modo in cui sono gestiti gli eventi. Il DOM ha due modalità di gestione: quella vecchia permette di associare un ascoltatore utilizzando una delle proprietà dell'elemento; quella nuova invece utilizza addEventListener() e removeEventListener(). Dart semplifica ulteriormente

rimuovendo tutte le proprietà di Element e creando una classe ElementEvents. Per ogni tipo di evento conosciuto esiste una proprietà in quella classe: click, mouseDown, etc. Ognuna di queste propriet è un oggetto evento che può aggiungere e rimuovere ascoltatori e gestire eventi:

```
// Old:
elem.addEventListener('click',(event) => print('click!'),false);
elem.removeEventListener('click', listener);
// New:
elem.on.click.add((event) => print('click!'));
elem.on.click.remove(listener);
```

3.7.3 Interazione tra Dart e il DOM

L'esempio Sunflower utilizza l'elemento Canvas di HTML5 per disegnare una immagine che assomiglia a un girasole. Il modo più semplice è di utilizzare direttamente HTML:

```
<html>
  <head>
    <title>sunflower</title>
    <link type="text/css" rel="stylesheet" href="sunflower.css">
  </head>
  <body>
    <h1>drfibonacci's* Sunflower Spectacular</h1>
    <h2>* Count the clockwise and counter-clockwise
                                    spiral arms at the edges</h2>
    <div class="image"></div>
    <div>
      <canvas id="canvas" width="300" height="300"></canvas>
    </div>
    <div>
      <label># seeds (n):</label>
    </div>
    <div class="rangeOption">
      <input id="slider" type="range" max="1000" value="0" />
      <label class="minLabel">0</label>
      <label class="maxLabel">1000</label>
    </div>
    <script type="application/dart" src="Sunflower.dart"></script>
  </body>
</html>
```

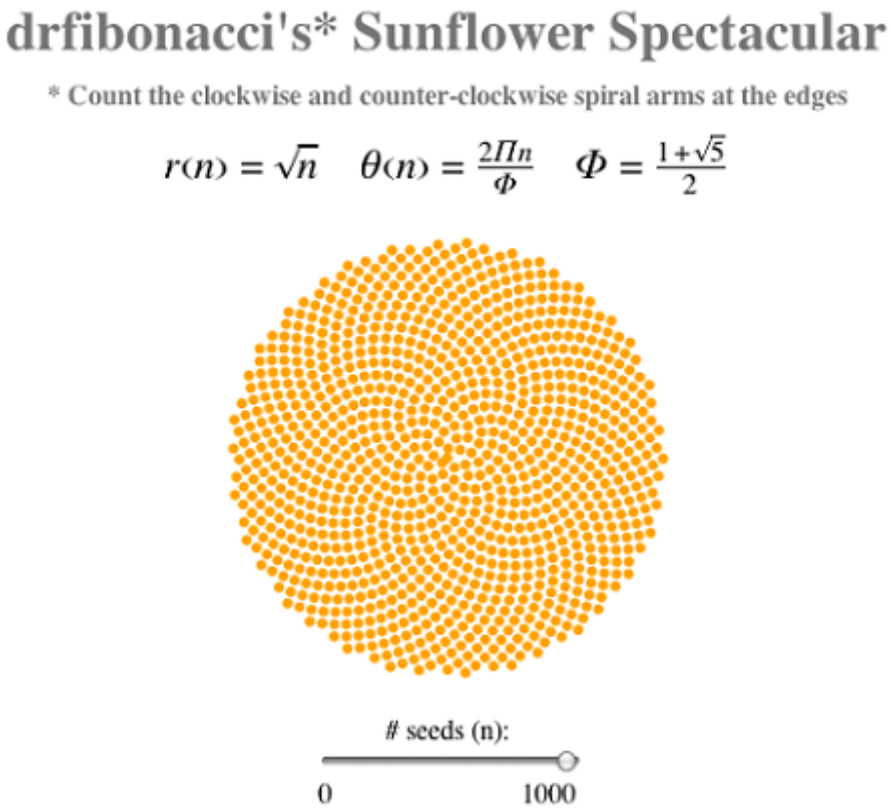
Il tag script include un sorgente di tipo application/dart, cioè lo script può essere eseguito in ogni browser che supporti Dart. Il codice Dart è il seguente:

```
// Definizione della classe Sunflower
class Sunflower {

  // Funzione principale invocata al caricamento
  static void main() {
    new Sunflower();
  }

  // Costanti
  static final SEED_RADIUS = 2;
  static final SCALE_FACTOR = 4;
  static final PI2 = Math.PI * 2;
  static final PHI = (Math.sqrt(5)+1) / 2;
  static final MAX_D = 300;
  static final ORANGE = "orange";

  // Variabili
  CanvasRenderingContext2D ctx;
  num xc, yc;
  num seeds = 0;

  // Costruttore
  Sunflower() {
    // Recupero degli elementi contenuti nella pagina
    Document doc = window.document;
    HTMLCanvasElement canvas = doc.getElementById("canvas");
    xc = yc = MAX_D / 2;
    ctx = canvas.getContext("2d");
    HTMLInputElement slider = doc.getElementById("slider");
    slider.onchange =(Event e) {
      seeds = Math.parseInt(slider.value);
      drawFrame();
    };
    drawFrame();
  }

  // Disegna la figura
  void drawFrame() {
    ctx.clearRect(0, 0, MAX_D, MAX_D);
    for (int i=0; i<seeds; i++) {
      num theta = i * PI2 / PHI;
      num r = Math.sqrt(i) * SCALE_FACTOR;
      num x = xc + r * Math.cos(theta);
```

```
      num y = yc - r * Math.sin(theta);
      drawSeed(x,y);
    }
  }
```

//Disegna un cerchio che rappresenta il seme a coordinate (X,Y)

```
  void drawSeed(num x, num y) {
    ctx.beginPath();
    ctx.setLineWidth(2);
    ctx.setFillColor(ORANGE);
    ctx.setStrokeColor(ORANGE);
    ctx.arc(x, y, SEED_RADIUS, 0, PI2, false);
    ctx.fill();
    ctx.closePath();
    ctx.stroke();
  }
}
```



Così come per ogni altro linguagio object-oriented, l'unità principale è una classe. Per convenzione i nomi delle classi terminano con '.dart'. L'esecuzione comincia con la funzione main() senza argomenti, che invoca il costruttore, il quale imposta Canvas, registra un ascoltatore di eventi e disegna il primo frame.

Dart fornisce dei wrapper per accedere agli elementi del DOM che possiamo utilizzare per inserire elementi nella pagina. Il costruttore richiama window.document.getElementById() per recuperare Canvas e gli oggetti slider utilizzando i loro ID. Il metodo drawFrame() esegue i calcoli mentre drawSeed() disegna su Canvas.

Combinando il layout HTML con un linguaggio a tipizzazione dinamico, Dart rende possibile creare facilmente anche complesse interfacce utente.

3.7.4 Client e server

Una delle caratteristiche principali di Dart è di essere un linguaggio sia client che server side. Nel seguente esempio verrà illustrata la struttura di una semplice applicazione client/server. Il codice della pagina HTML è minimale poichè referenzia solamente il codice Dart:

```
<html>
  <head>
  </head>
  <body>
    <script type="application/dart"src="client.dart"></script>
  </body>
</html>
```

Il codice di Dart lato client sarà fatto come seguente:

```
//client.dart
#library("client");

#import("dart:html");
#import("dart:json");
#import("dart:core");

void main() {
  InputElement forename = document.createElement("input");
  forename.value = "Chris";
  window.document.body.nodes.add(forename);

  InputElement surname = document.createElement("input");
  surname.value = "Buckett";
```

```
  window.document.body.nodes.add(surname);

  DivElement output = document.createElement("div");

  ButtonElement submit = document.createElement("button");
  submit.text = "Submit";
  submit.on.click.add((event) {
    //the event handler function
    sayHelloFromServer(forename.value, surname.value,
                                    (String messageText) {
      //the callback function
      output.innerHTML += messageText + "<br/>";
    });

  });

  window.document.body.nodes.add(submit);
  window.document.body.nodes.add(output);

}

void sayHelloFromServer(String forename,String surname,callback){
  XMLHttpRequest req = new XMLHttpRequest();
  String url = "http://localhost:9090/app/greet";
  req.open("POST",url,false);

  req.on.load.add((event) { //
    window.console.log("response recieved");
    String json = req.responseText;
    window.console.log(json);
    Map<String,String> result = JSON.parse(json);
    String serverMessage = result['serverMessage'];
    String messageText = "\${serverMessage}
                \${result['forename']} \${result['surname']}";
    window.console.log("calling callback");
    callback(messageText);
  });
  String data = '{"forename":"\${forename}",
                                      "surname":"\${surname}"}';
  window.console.log("sending data to server: \${data}");
  req.send(data);
}
```

La prima parte consiste degli import delle libreire. La funzione main() costrusice l'interfaccia utente creando alcuni elementi di input, un bottone submit e un div di output.

Come gestore dell'evento di pressione del bottone viene usata una funzione anonima. Quando il bottone viene premuto, passiamo il valore del nome e il valore del cognome delle textbox alla funzione $sayHelloFromServer(forename, surname, callback)$ passando anche una funzione anonima come funzione di callback. Questa verrà chiamata alla ricezione dei dati di risposta dal server.

La funzion $sayHelloFromServer()$ crea un XMLHttpRequest che esegue una POST a $http://localhost:9090/app/greet$. I dati sono costruiti come una stringa json e poi chiamiamo $req.send(data)$ per inviare i dati al server.

Quando il server risponde viene chiamato il gestore di evento req.on.load, che recupera i dati rilevanti e li passa alla funzione callback. La callback aggiunge il messaggio al div di output visualizzabile nell'interfaccia utente. Il codice lato server invece:

```
#library("myserver");

#import("dart:json");
#import("/home/myPath/fling.dart");

void main() {
  HttpServer server = new HttpServer();

  //serve the static pages from the client
  server.handle("/client/", ClientApp.create("../client/"));

  //handle requests for data
  server.handle("/app/greet",(HttpRequest req,HttpResponse res){
    res.setHeader("Content-Type", "text/plain");

    if (req.method == "POST") {

      print("Received data: \${req.body}");
      Map<String,String> input = JSON.parse(req.body);
      String fname = input['forename'];
      String sname = input['surname'];
      print("Parsed data: \${fname} \${sname}");
      String message = "Server says, Hi there";

      String output = '{"forename":"\${fname}","surname":
                    "\${sname}","serverMessage":"\${message}"}';
      res.write(output);
      print("Sent Data \${output}");

    }
```

```
    else {
      res.write("POST only please");
    }

    //close the response
    res.finish();
  });

  server.listen(9090);
  print("listening on port 9090");

  // Runs the message loop.
  Fling.goForth();

}
```

Gli import sono richiesti per rendere disponibili gli oggetti HttpServer, HttpRequest, HttpResponse e ClientApp. Il main() crea HttpServer e prepara due gestori di evento. Il primo gestisce la richiesta di pagine statiche da parte del client, il secondo invece gestisce la richiesta di dati. Il secondo in particolare è in grado di gestire un oggetto HttpRequest e uno HttpResponse. Se il metodo utilizzato è POST allora estraiamo i dati usando req.body. Dopodichè vengono creati dei messaggi di log nella console e restituiamo un saluto dal server: $''`Server says, Hi there'''$. La chiamata server.listen(9090) posiziona il server in ascolto sulla porta 9090.

3.8 Approccio al web

Dart ha attirato su di se molte attenzioni da parte degli addetti ai lavori, anche se la sua penetrazione è comprensibilmente lenta. Per promuovelo sono in atto diverse iniziative:

- Dart Synonym

  La maggior parte degli sviluppatori web possiede una buona conoscenza di Javascript, partendo da questa considerazione due sviluppatori Google hanno pubblicato Dart Synonym, una pagina che fornisce le traduzioni di una serie di istruzioni Javascript nel linguaggio Dart. Questa iniziativa è stata presa allo scopo di semplificare l'apprendimento di Dart agli sviluppatori Javascript. Altri esempi si possono trovare nel sito ufficiale.

Getting started

Code embedding

```
<script src='program.js'></script>
```

```
// Note: This will only work in Dartium (a build of
// Chromium with Dart VM)
<script type='application/dart' src='program.dart'></script>

// Also, you'll need this to kickstart the Dart engine.
<script type='text/javascript'>
  if (navigator.webkitStartDart) {
    navigator.webkitStartDart();
  }
</script>
```

Entry point

```
// Not required.
function main() {
  // To be used as the entry point, but it must be
  // called manually.
}

main();

// Sometimes the entry point is written as an
// anonymous function
(function(){
  // Code to be run automatically on execution
})();
```

```
// REQUIRED.
main() {
  // this is the entry point to the program
}
```

Variables

Create + assign value

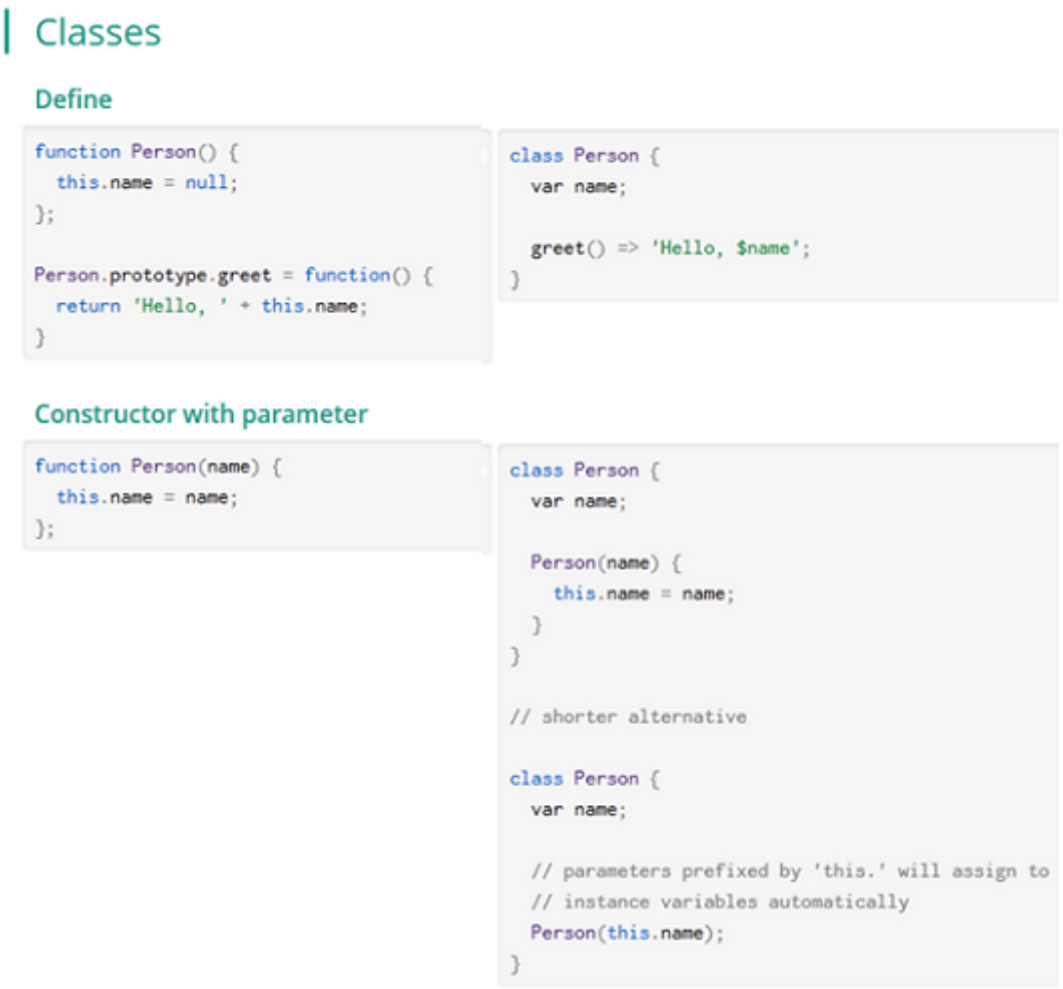
```
var myName = 'Aaron';
```

```
// Dart variables can be typed...
String myName = 'Aaron';

// but they don't need to be
var myOtherName = 'Aaron';
```

Default value

```
var myName;
// == undefined
```

```
var myName;
// == null

int x;
// == null
```



Classes

Define

```
function Person() {
  this.name = null;
};

Person.prototype.greet = function() {
  return 'Hello, ' + this.name;
}
```

```
class Person {
  var name;

  greet() => 'Hello, $name';
}
```

Constructor with parameter

```
function Person(name) {
  this.name = name;
};
```

```
class Person {
  var name;

  Person(name) {
    this.name = name;
  }
}

// shorter alternative

class Person {
  var name;

  // parameters prefixed by 'this.' will assign to
  // instance variables automatically
  Person(this.name);
}
```

- DARTBox2D

  è un porting di Box2D, il motore fisico open source di Angry Birds, effettuato con il nuovo linguaggio di Google. L'intenzione della società è dimostrare la possibilità di creare dei videogiochi in DART. Box2D è un motore fisico 2D scritto in C++, sviluppato inizialmente da due stagisti di Google, che ha avuto piuttosto successo, tanto da essere stato usato, tra le altre cose per il gioco Crayon Physics Deluxe, vincitore del Independant Game Festival Grand Prize nel 2008. Il motore è disponibile, oltre che in C++, anche per Flash, Java, C, Python e ora anche Dart.

- EclipseCon 2012

  Nell'edizione 2012 dell'EclipseCon, l'annuale conferenza della Eclipse Community che si terrà in Virginia a fine marzo, verrà dedicato ampio spazio a Dart. Un'intera sessione, intitolata Dart in Action, si occuperà di fornire supporto a questo nuovo linguaggio, il responsabile sarà Dan Rubel, di Google, che sta lavorando a un editor Dart basato su Eclipse. Sul sito del linguaggio Dart è presente un FAQ che spiega: 'Google sta impegnando numerose risorse dietro sia Dart sia Javascript; la scelta di sviluppare Dart ma allo stesso tempo usare Javascript estensivamente, lavorando su tool, implementazioni e specifiche di linguaggio per Javascript, è stata fatta perchè pensiamo che Dart lo valga'.

Capitolo 4

Conclusioni

Ormai è chiaro che nelle scienze informatiche assistiamo ad una evoluzione incredibilmente rapida, sia per gli sviluppi teconologici, sia per l'introduzione di nuovi paradigmi concettuali.

Il problema di una evoluzione così rapida, sta nel fatto che queste innovazioni fanno spesso fatica ad essere accettate e utilizzate da coloro che fino a quel momento hanno fatto affidamento su ciò che avevano a loro disposizione, un po' per una diffidenza innata in ciò che è poco conosciuto, un po' per questioni di legacy tecnologico.

A monte del linguaggio Dart ci sono tutte quelle considerazioni sulle problematiche che riguardano il Web moderno che ai linguaggi attuali mancano; in particolare l'incorporazione al proprio interno di funzionlità che in questo momento occorre gestire utilizzando differenti tipi di tecnologie, che non garantiscono n` il pieno funzionamento nel caso di una integrazione le une con le altre, nè il medesimo risultato di esecuzione in ambienti differenti.

L'introduzione di un sistema basato sugli oggetti garantisce le proprietà strutturali che sono richieste, ormai, per qualsiasi linguaggio, mentre la possibilità di una tipizzazione dinamica permette un elevato grado di compatibilità con i linguaggi che lo hanno preceduto, in particolar modo con Javascript, a cui Dart è legato in maniera molto stretta.

Dart dovrebbe risultare, quindi, semplice e familiare a entrambi gli opposti schieramenti di coloro che vengono dai linguaggi per il web e di coloro che vengono dai linguaggi strutturati ad oggetti; i primi, infatti, tendono a percepire qualsiasi definizione strutturale formale come una limitazione della propria libertà espressiva, mentre i secondi vorrebbero poter sempre sfruttare i meccanismi propri dei paradigmi ad oggetti, quali ereditarietà e polimofismo. Dart accontenterebbe entrambi, garantendo sia la possibilità di appoggiarsi ad una forte struttura object oriented, sia la più completa libertà espressiva di un

linguaggio non tipato, a completa discrezione del programmatore.

Questa grande varietà di paradigmi di programmazione applicabili, porta però anche a degli effetti che possono risultare spiacevoli. Degli esempi di ciò possono essere la possibilità di definire identificatori di proprietà con delle parentesi tonde vuote alla fine, rendendole, di fatto, indistinguibili da un metodo senza argomenti; oppure il fatto che il meccanismo di type checking, qualora si utilizzi una tipizzazione debole, si accorga dell'incompatibilità dei tipi utilizzati solo alla prima invocazioni del metodo o della funzione interessata, risultando in un errore runtime anche molto, ma molto tempo dopo il lancio dell'applicazione.

In conclusione il linguaggio Dart, anche se, essendo ancora in una fase iniziale di sviluppo non è esente da difetti, presenta molte caratteristiche che lo rendono un buon candidato tra le tecnologie che concorrono ad affermarsi all'interno del Web 2.0.

Capitolo 5

Bibliografia

http://blog.chromium.org/
http://www.dartlang.org/
http://googlecode.blogspot.com/
http://blog.html.it/
http://www.html.it/
http://www.googlab.it/
http://www.w3.org/
http://www.juliusdesign.net/
http://www.html5today.it/
http://dartinside.com/
http://video.pmi.it/
http://programmazione.it
http://www.html5italia.com/
http://xhtml.com/en/xhtml/reference/
http://www.html.it/
http://www.7thfloor.it/
http://michelebologna.net/
http://www.free-os.it/
http://ars-informatica.blogspot.com/
http://tiggilibero.altervista.org/
http://www.extrowebsite.com/
http://dartwatch.com/
http://www.greatandlittle.com/

http://www.sun.com
http://www.superbanner.org/blog/
http://www.fog.it/
http://www.yourinspirationweb.com/
http://devteam.vivido.it/
http://www.ibm.com/us/en/
http://xhtml.html.it/
http://www.webcreazionesiti.com/
http://www.wordpresstutorial.it/
http://news.pmiservizi.it/
http://mirkoagrati.blogspot.com/
http://www.ideativi.it/
http://www.antoniopicone.it/
it.wikipedia.org/
www.ossblog.it