ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA SEDE DI CESENA

Seconda Facoltà di Ingegneria con Sede a Cesena
Corso di Laurea in Ingegneria Informatica

SVILUPPO DI APPLICAZIONI WEB STRUTTURATE: DART E JAVASCRIPT A CONFRONTO

Elaborata nel corso di:
Sistemi Distribuiti

Tesi di Laurea di:
MARCHI FRANCESCA

Relatore:
Prof. ALESSANDRO RICCI

SESSIONE III
ANNO ACCADEMICO 2010-2011

PAROLE CHIAVE

Web

Html5

JavaScript

Dart

Structured Web Programming

A Mauro e Flavia
a cui devo quello che sono.
A Francesco
per il suo amore.
A tutti coloro
che mi hanno sempre
sostenuto e hanno creduto in me.

Indice

Introduzione

Negli ultimi anni il Web ha assunto un ruolo, di giorno in giorno, sempre più centrale. La presenza sempre più diffusa e disponibile di una connessione ad Internet ad alta velocità, ha portato il Web ovunque, anche in luoghi dove, pochi anni fa, non si sarebbe mai immaginato.

La realizzazione e la diffuzione, in continua crescita, di applicazioni Web con interfacce grafiche sempre più accattivanti e funzionalità sempre più avanzate hanno portato ad un utilizzo del Web sempre più pervasivo, sia all'interno degli ambienti lavorativi che all'interno delle mura domestiche.

Per arrivare a dove si è oggi, il Web ha subito notevoli cambiamenti. La prima vera evoluzione si ebbe, quando alla fine degli anni '90, si passò dal concetto di Web 1.0 all'evoluto Web 2.0.

Il Web 1.0, non prevedeva il concetto di interazione, era concepito solamente come un modo per visualizzare i documenti ipertestuali, molto differente dall'idea di base del Web 2.0, il quale mise in evidenza la necessità di modificare l'approccio con cui gli utenti si rivolgevano al Web. Esso introdusse il concetto di interazione fra il Web e gli utenti che ne fanno uso. Gli utenti passarono, perciò, dall'essere semplici utenti "passivi", a coloro che rivestono un ruolo centrale nel processo di partecipazione al Web, cioè veri e proprio utenti "attivi".

Nell'evoluzione del Web, ebbe un ruolo particolarmente importate, l'introduzione dei linguaggi di scripting, come JavaScript. I linguaggi si scripting sono sempre esistiti, anche prima della loro introduzione nelle pagine Web. Gli script realizzati con tali linguaggi avevano lo scopo di far interagire altri programmi fra loro.

Quando fu introdotto, all'interno delle pagine Web, il linguaggio di scripting JavaScript, trasformò normali pagine Web statiche in pagine Web dinamiche, con le quali un utente poteva interagire per mezzo di bottoni, campi

di testo, forms, etc. L'introduzione di JavaScript, perciò, ha modificato molto il modo di vedere e di utilizzo del Web.

Con il tempo e l'utilizzo sempre più persistente dei servizi offerti dal Web, è emerso il concetto Software as a Service. Tale concetto rappresenta il modello software che sta alla base delle applicazioni Web ora presenti sul Web.

Le applicazioni Web sono dei veri e propri sistemi software, i quali si discostano molto dal modello dei software proprietari, perché essi non risiedono più direttamente sugli elaboratori degli utenti, ma sono applicazioni disponibile tramite il Web e utilizzabili mediante l'uso di un semplice browser Web.

Il modello Software as a Service, ha portato le tecniche che stanno alla base del Web, come JavaScript e Html, ad evolversi per supportare tutte quelle caratteristiche che le applicazioni Web devono avere per poter competere con i "vecchi" software proprietari.

Il W3C per rimanere al passo con le esigenze, ha cominciato a lavorare su una nuova specifica Html. La nuova specifica Html5 in sviluppo, ha aggiunto nuove Web API per introdurre tutte quelle funzionalità rivolte alla realizzazione delle applicazioni Web.

ECMAScript, negli anni, ha revisionato il linguaggio di scripting JavaScript per renderlo, non più un semplice linguaggio per la creazione di script, ma un vero e proprio linguaggio di programmazione Web lato client.

Date le esigenze nate negli ultimi tempi, oltre ad adattare linguaggi di programmazione già esistenti, ne sono nati di nuovi. La nascita di questi nuovi linguaggi è una diretta conseguenza di come si sta evolvendo il Web. Uno degli ultimi linguaggi di programmazione Web sviluppati è Dart. Esso è nato con l'obbiettivo di fornire un linguaggio di programmazione Web strutturata in grado di far fronte a quelle lacune che linguaggi già presenti, come JavaScript, può presentare nella programmazione Web.

L'obbiettivo della presente tesi è quello di mostrare quali sono le differenza, in campo di programmazione Web, tra il nuovo linguaggio Dart, rispetto al "veterano" e ormai consolidato JavaScript. Tale distinzione viene effettuata nell'ottica della progettazione e realizzazione di applicazioni Web basate sul modello di Software as a Service.

Nel primo capitolo viene mostrata l'evoluzione del Web, a partire dalle origini fino ad arrivare al Cloud Computing. Viene introdotta la tecnologia che sta alla base del Web, ovvero HTML, soffermandosi più nel dettaglio

sulla nuova specifica Html5 che si sta sviluppando al momento, a descrivendo le nuove funzionalità offerte nell'ottica dell'utilizzo delle applicazioni Web.

Nel secondo capitolo viene descritto nel dettaglio il linguaggio di scripting JavaScript, concentrandosi sugli aspetti più interessanti del linguaggio, come l'object-oriented, l'interfacciamento con il DOM, la struttura e gli importantissimi eventi.

Nel terzo capitolo, viene presentato il nuovo linguaggio di programmazione Web strutturata Dart. Tale presentazione viene mostrata nell'ottica delle somiglianze o delle differenze che lo caratterizzano da JavaScript, soffermandosi su argomenti come interfacce e classi, tipi opzionali, eventi e concorrenza.

Infine, nel quarto capitolo viene fornito un quadro sintetico di ciò che è emerso dal confronto fra i due linguaggi e vengono espresse alcune considerazioni conclusive in merito alla programmazione di applicazioni Web.

Capitolo 1

World Wide Web e Html

1.1 Evoluzione del Web

La maggior parte delle persone confonde il World Wide Web con Internet. Anche se legati, Internet e World Wide Web, sono due concetti diversi.

Internet è una rete di computer mondiale ad accesso pubblico, attualmente rappresentante il principale mezzo di comunicazione di massa, che offre all'utente una vasta scelta di contenuti e servizi.

Tale connessione è resa possibili per mezzo di una suite di protocolli di rete standard chiamata "TCP/IP", dal nome dei suoi due principali protocolli, "TCP" e "IP", che comprendono le regole comuni con cui i computer connessi a Internet comunicano tra loro, indipendentemente dalla loro architettura hardware e software.

Internet offre i più svariati servizi, ed è utilizzato per le comunicazioni più disparate: private e pubbliche, lavorative e ricreative, scientifiche e commerciali. I suoi utenti, in costante crescita, nel 2008 hanno raggiunto quote 1,5 miliardi e, visto l'attuale ritmo di crescita, si prevede che saliranno a 2,2 miliardi nel 2013.

Allora, il World Wide Web, che cos'è?

Il Word Wide Web, abbreviato come WWW, e comunemente noto come il Web, è il principale servizio offerto da Internet che permette di visualizzare ed usufruire di un insieme vastissimo di testi, immagini, video, contenuti multimediali, e molto altro.

Tutto ha inizio nel 1989 quando lo scienziato informatico Tim Berners-Lee, al laboratorio europeo per la fisica delle particelle (CERN) di Ginevra,

propose una rete di documenti connessi tramite collegamenti ipertestuali ospitati da computer, chiamati server ipertestuali.

Le intenzioni erano quelle di soddisfare l'esigenza degli scienziati, di una più efficiente condivisione delle informazioni tra i membri della comunità scientifica dell'istituto per cui lavorava.

Cominciò a lavorare sul progetto, con l'obbiettivo di proporre un sistema Internet basato sugli ipertesti.

Berners-Lee creò un'applicazione del linguaggio SGML (Standard Generalised Markup Language) e lo chiamò HTML (HyperText Markup Language).

Il "debutto" ufficiale del Web come servizio Internet risale al 6 Agosto del 1991, giorno in cui Berners-Lee mise su Internet il primo sito Web.

Oltre alla pubblicazione del primo sito Web, Tim Berners-Lee decise di rendere pubblica la tecnologia che c'era alla base del Web e pubblicò la prima descrizione pubblica dell'HTML, chiamato "HTML Tags". In questo documento vennero descritti i primi 20 elementi che costituivano l'iniziale e relativamente semplice progetto dell'HTML. Tredici di questi elementi sono ancora oggi presenti nell'HTML4.

Nel 1993, con l'introduzione del Web browser "Mosaic", si ebbe un punto di svolta per il World Wide Web. "Mosaic", sviluppato dal team "National Center for Supercomputing Applications" presso l'Università dell'Illinois, fu il primo Web browser grafico. Prima del suo avvento, la grafica non era integrata con il testo contenuto nelle pagine Web. La grafica dell'interfaccia utente promossa da "Mosaic" permise al Web di diventare, di gran lunga, il servizio Internet più popolare.

Nel 1994 l'IETF (Internet Engineering Task Forse) creò l'HTML Working Group, che nel 1995 completò "HTML 2.0", la prima specifica HTML destinata ad essere trattata come uno standard dalle future implementazioni. La denominazione "2.0" fu introdotta per distinguere la versione aggiornata dalle versioni precedenti.

Dal 1996 le specifiche HTML sono state assunte dal World Wide Web Consortium (W3C) che negli anni a seguire pubblicò diverse specifiche HTML fino ad arrivare al 1999 all'ultima pubblicazione ufficiale: HTML 4.01.

Data la diffusione e l'utilizzo, nel 2000 la specifica HTML 4.01 venne standardizzata (ISO/IEC 15445:2000) e da allora rappresenta la versione ufficiale più recente.

Dopo la standardizzazione della specifica HTML 4.01, HTML Working Group del W3C si occupò di un linguaggio parallelo, basato sull'XML, XHTML, linguaggio che associa alcune delle proprietà XML con le caratteristiche HTML. Tale linguaggio prevede un uso più restrittivo dei tag HTML sia in termini di validità che in termini di sintassi per la descrizione della struttura logica della pagina.

Fin dalla pubblicazioni delle prime specifiche si manifestò un immediato e ampio successo del Web, in virtù della possibilità per chiunque di diventare editore.

Con il successo del Web ha inizio la crescita esponenziale e inarrestabile di Internet e dei suoi servizi, ancora oggi in atto, nonché della cosiddetta "era del Web".

L'"era del Web" è caratterizzata da fasi ben stabilite, dovute all'evoluzioni delle tecnologie e alle esigenze emerse dall'utilizzo del Web.

La prima fase, denominata con molta fantasia **Web 1.0**, descrive un World Wide Web come un insieme di pagine Web collegate tra loro mediante collegamenti ipertestuali.

Il Web 1.0 degli anni '90, presentava una versione statica dei siti Internet, ben diversa da quella attuale. L'utente poteva solo "navigare" tra i vari siti, per mezzo dei collegamenti ipertestuali, senza interagire con essi.

Il web era concepito solamente come un modo per visualizzare documenti ipertestuali in formato Html, il quale rendeva testo e contenuto inseparabili. L'utente, quindi, era un "navigatore passivo", cioè colui che entra in una pagina web, ne legge il contenuto e se ne va, proprio come leggere un libro o una enciclopedia.

L'interazione con il Web, era un concetto che non faceva parte del Web 1.0.

In poco più di un decennio, date le esigenze e le lacune del Web, si passò dal Web 1.0 al **Web 2.0**.

Con il termine Web 2.0, non si voleva introdurre una nuova tecnologia, ma piuttosto una evoluzione del precedente Web 1.0, caratterizzato da siti Web statici, senza la possibilità di interazione, ad un nuovo modo di vedere il Web.

Il termine Web 2.0 racchiude in sé l'insieme di tutte quelle applicazioni online che permettono uno spicco livello di interazione tra il sito e l'utente (blog, forum, chat, wiki, youtube, faceboo, myspace, twitter, google+, wordpress).

Grazie all'integrazione con i database, all'utilizzo di linguaggi di scripting come JavaScript, degli elementi dinamici e dei CSS per gli aspetti grafici, si possono creare vere e proprie "applicazioni Web" che si discostano parecchio dal vecchio concetto di semplice ipertesto e che puntano ad assomigliare ad applicazioni tradizionali per computer.

L'introduzione degli scripting language, ha segnato un passo importante nell'evoluzione del Web.

I linguaggi di script sono linguaggi di programmazione interpretati, destinati in genere a compiti di automazione oppure utilizzati all'interno di pagine Web. Inizialmente i primi linguaggi di scripting nacquero dall'esigenza di automatizzare alcune operazioni di programmazione e interazione. Poi con lo sviluppo del Web, gli scripting furono integrati all'interno delle pagine Web.

L'integrazione dei linguaggi di scripting all'interno delle pagine Web ha portano un notevole cambiamento nell'utilizzo del Web. Le pagine non rappresentavano più semplici documenti per la visualizzazione del contenuto, ma divennero vere e proprie interfacce attraverso le quali l'utente poteva interagire con il Web.

Da un punto di vista strettamente tecnologico, il Web 2.0 è del tutto equivalente al Web 1.0, in quanto l'infrastruttura di rete continua ad essere regolata dal TCP/IP e HTTP, e l'ipertesto è ancora il concetto base delle relazioni fra i contenuti.

Gli aspetti di maggior interesse che differenziano il Web 2.0, dal suo predecessore Web 1.0, sono:

- l'approccio con il quale gli utenti si rivolgono al Web: si passa fondamentalmente dalla semplice consultazione, cioè dall'essere un utente "passivo", all'essere un utente "attivo", cioè colui che riveste un ruolo centrale nel processo di partecipazione alla crescita del Web, con contenuti e capacità proprie;

- l'utilizzo del Web come piattaforma: passaggio dall'utilizzo del computer desktop, all'utilizzo del Web come piattaforma, fornendo l'accesso mediante i singoli browser.

Questo ultimo aspetto, cioè l'utilizzo del Web come piattaforma, ha portato allo sviluppo di un modello che ha rivoluzionato il modo di vedere il Web: il **cloud computing**. (Vedi Figure 1.1)

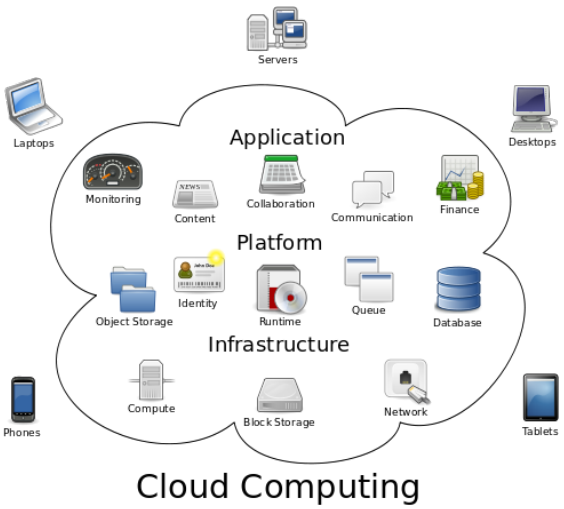

Figura 1.1: Cloud Computing

Il cloud computing indica un insieme di tecnologie che permettono, tipicamente sotto forma di un servizio offerto da un provider al cliente, di memorizzare/archiviare ed elaborare dati grazie all'utilizzo di risorse hardware/software distribuite e virtualizzate in rete.

All'interno del cloud computing è possibile distinguere tre tipologie fondamentali di servizi (Vedi Figure 1.2):

- Software as a Service (SaaS): utilizzo di software in remoto, spesso attraverso un server web;
- Platform as a Service (PaaS): utilizzo di una piattaforma remota come ambiente di esecuzione e come database;
- Infrastructure as a Service (IaaS): utilizzo di risorse hardware in remoto.

La premessa che sta alla base del cloud computing consiste nell'assumere che in questo nuovo modo di vedere il Web i servizi hardware a software devono risiedere sui server web, e non più sui singoli dispositivi connessi alla rete. Lo scenario è quindi quello di un utente, avente a disposizione

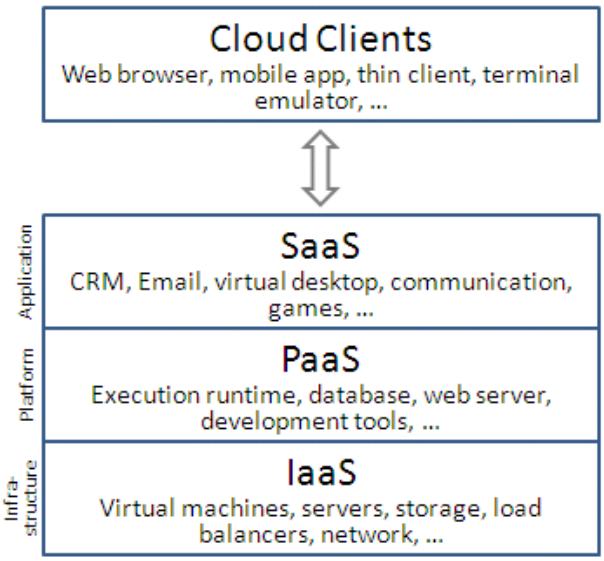

Figura 1.2: Cloud Computing Layers

un browser e una connessione ad Internet, che può accedere ai vari servizi offerti dal cloud.

Fra le varie tipologie offerte dal cloud computing, ve ne una che negli ultimi tempi, con la diffusione ormai ovunque di Internet e con l'aumento delle esigenze degli utenti ha un ruolo particolarmente importante, si tratta del **Software as a Service**.

Il Software as a Service rappresenta un modello software distribuito, nel quale il software ed i relativi dati sono ospitati centralmente nella "nuvola" di Internet, e sono in genere accessibili agli utenti tramite l'utilizzo di un semplice browser Internet.

Non si ha più a che fare con i "vecchi" software proprietarie, installati sulle macchine degli utenti, ma ora si a che vedere con le **applicazioni Web**, la diretta conseguenza del modello Software as a Service.

Le applicazioni Web, o anche Web Application, rappresentano tutte quelle applicazioni web-based distribuite, cioè applicazioni che in pratica non risiedono più direttamente sulle macchine che le usano, ma su elaboratori remoti, come server.

Quali sono i vantaggi nell'utilizzare una Web Application piuttosto che

un software tradizionale?

- nessun software da scaricare e installare sulla computer desktop;
- nessuna necessità di upgrading del software quando vengono aggiunte nuove funzionalità o eliminati problemi esistenti;
- è sufficiente un "thin client" con minimi requisiti hardware;
- utilizzabile da qualsiasi elaboratore, purché questo abbia a disposizione una connessione Internet (negli ultimi tempi sono state studiate tecniche per l'utilizzo del Web Application anche offline);
- in caso si trattino documenti, questi possono essere condivisi ed accessibili da chiunque ne abbia i diritti;
- non è necessario l'acquisto della licenza software, basta un semplice abbonamento qualora il servizio sia a pagamento;
- tutti i dati risiedono su un server remoto, non gestibile direttamente.

Con la possibilità di creare delle vere applicazioni Web con il preciso scopo di andare a sostituire i software tradizionali, o anche detti software proprietari, sono emerse tutta quella serie di caratteristiche che in precedenza appartenevano ai software proprietari, che ora devono essere integrate all'interno delle applicazioni Web.

Cambiando le esigenze del Web, di conseguenza sono cambiate le tecnologie che ne stanno alla base.

Ad oggi, quindi, il Web deve essere in grado di fornire tutte quelle tecnologie per lo sviluppo di veri e proprio software web.

Per esempio, i linguaggi di scripting come JavaScript, che fino a qualche hanno fa, venivano utilizzati per rendere le pagine interattive agli occhi degli utenti, oggi sono utilizzati come veri e propri linguaggi di programmazione di software.

Il browser non è più solo utilizzato come semplice interprete di pagine web, ma viene utilizzata come piattaforma di elaborazione per le applicazioni web.

Tali tecnologie hanno dovuto adeguarsi al cambiamento del Web. Hanno dovuto apportare tutte quelle modifiche che il Web ha richiesto. Oltre alle modifiche delle tecnologie già esistenti, ne sono nate di nuove, pronte a soddisfare tutte le richieste del nuovo modello del Web.

1.2 Cos'è l'Html

Il formato più comune, oltre al più conosciuto, per la pubblicazione di documenti nel web e la creazione di applicazioni è l'HTML.

Dalla sua comparsa esso è stato definito come un linguaggio semplice, concepito principalmente per la descrizione dei documenti. Esso è poi cresciuto ed è stato adattato ad una vasta gamma differente che va dalla pubblicazione di notizie ai blog, fino ad arrivare alle applicazioni per e-mail, mappe, elaborazione di testi e fogli di lavoro.

HTML è l'acronimo di HyperText Markup Language e permette di descrivere i documenti ipertestuali disponibili nel Word Wide Web.

Alla base del World Wide Web ci sono gli ipertesti, insieme di documenti messi in relazione tra loro per mezzo di collegamenti ipertestuali, visualizzabili tramite browser.

Un HyperText Markup Language non è un linguaggio di programmazione, perché non prevede il concetto di variabile e di funzione, ma è un linguaggio di markup che i web browsers usano per interpretare e comporre testo, immagini e altro materiale da visualizzare su web pages.

Un linguaggio di markup è un insieme di regole che descrivono i meccanismi di rappresentazione dei dati, esso è composto da:

- tag o marcatori, elementi di sintassi racchiusi tra due parentesi angolari che consistono nelle istruzioni che rappresentano le caratteristiche del documento testuale;
- una grammatica che stabilisce le regole formali di utilizzo dei tag;
- infine da una semantica che definisce il dominio di applicazione e la funzione dei tag.

Tradizionalmente i linguaggi di markup sono divisi in due tipologie:

- linguaggi di markup di tipo procedurale, dove vengono specificate le procedura che devono essere eseguite affinché un testo assuma una determinata presentazione (es. Tex, LaTex, etc..);
- linguaggi di markup di tipo descrittivo, descrive la struttura di un componente identificandone i componenti (es. SGML, HTML, XML, etc..).

I linguaggi descrittivi risultano più vantaggiosi perché permettono di mantenere una netta separazione tra la struttura e la visualizzazione.

L'SGML (Standard Generalized Markup Language) è stato il primo metalinguaggio di markup descrittivo definito come Standard ISO. Principale funzione di SGML è la stesura di testi chiamati DTD (Document Type Definition), ciascuno dei quali definisce in modo rigoroso la struttura logica che devono avere i documenti di un determinato tipo; si dice che questi documenti rispetto a SGML costituiscono un linguaggio obiettivo, ovvero un'applicazione.

L'idea generale di SGML è quella di definire linguaggi di marcatura generica chiamata marcatura "descrittiva"; ogni linguaggio obiettivo definisce le caratteristiche strutturali dei documenti che governa.
L'organizzazione di un documento in un linguaggio obiettivo non si preoccupa della sua resa visiva, che potrebbe differire in base ai diversi dispositivi di visualizzazione che si utilizzano, ma piuttosto si preoccupa dei ruoli logico-semantici che rivestono le parti nelle quali il documento si articola, come ad esempio: paragrafi, capitoli, citazioni e tabelle.

Nonostante l'SGML è un linguaggio avente scopi più generici, l'HTML ne è una sua estensione. Infatti una delle caratteristiche importanti dell'HTML è che esso è stato concepito per definire il contenuto logico e non l'aspetto finale di un documento visualizzabile tramite il Web.

Ogni documento ipertestuale scritto in HTML deve essere contenuto in un file. Ogni file corrisponde ad una pagina web.

Le pagine web vengono inviate all'utente da un web server, cioè da un programma che si trova su un computer remoto, costantemente collegato alla rete Internet, e che per lo più non fa altro che inviare le pagine a chi ne fa richiesta.

Invece, lato utente, le pagine inviate vengono visualizzate sul browser, programma che permetter di leggere le pagine scritte in linguaggio HTML.

Attualmente i documenti HTML sono in grado di incorporare molto tecnologie diverse, che offrono la possibilità di aggiungere al documento ipertestuale tutte quelle caratteristiche non fanno parte del contenuto logico della pagina come la grafica, interazioni dinamiche con l'utente, animazioni interattive e contenuti multimediali. Si tratta di linguaggi come CSS per la formattazione dei documenti, JavaScript per rendere il documento dinamico, jQuery, XML, JSON, o di altre applicazioni per la gestione di

componenti multimediali come animazioni vettoriali o di streamig video o audio.

1.3 Verso nuovi orizzonti: HTML5

La panoramica del Web è cambiata molto dall'assunzione a W3C e dalla pubblicazione dell'ultima versione delle specifiche, avvenuta verso la fine del 1999.

In quel tempo il Web era strettamente legato al concetto di ipertesto e l'azione più comune per l'utente era l'utilizzo di contenuti tipicamente in formato testuale.

La mediamente bassa velocità di connessione e il limitato investimento sul media contribuivano ad una scarsa presenza di applicazioni Web, molto costose da sviluppare ed esigenti in termini di banda.

Man mano che la banda di connessione e che l'utilizzo dell'HTML aumentava, parallelamente aumentavano anche le esigenze e le richieste degli sviluppatori, facendo emergere sempre più i limiti di questo linguaggio.

Nel 2004, mentre il lavoro del W3C era incentrato sugli sviluppo futuri del XHTML2.0, il WHATWG (Web Hypertext Application Technology Working Group), un gruppo di lavoro esterno al W3C, ha iniziato a lavorare sul nuovo standard, non più aggiornato dal 2000, vedendolo come il prossimo passo per lo sviluppo dell'HTML.

L'intento iniziale è stato quello di proporre nuovi comandi e funzionalità fino ad allora ottenute in maniera non standard, ossia mediante il ricorso a plug-in o estensioni proprietarie.

Recentemente il W3C ha preso atto degli sviluppi ed ha cominciato a collaborare con i membri del WHATWG, mettendo momentaneamente da parte il gruppo XHTML2.0 preferendo di fatto la direzione intrapresa dall'HTML5.
Ad oggi è possibile trovare le specifiche sia nel sito del W3C che in quello che WHATWG, le quali sicuramente, in futuro, le vedremo fuse assieme in un'unica specifica. L'HTML5 è considerata la quinta specifica dello standard HTML, nello specifico si propone di diventare l'evoluzione dell'attuale HTML4.01.

1.4 Le novità dell'HTML5

Per prima cosa è importante ricordare che, anche in virtù della storia della sua nascita, all'interno dell'HTML5 convivono in armonia due anime: la prima, che raccoglie l'eredità semantica dell'XHTML2.0, e la seconda che invece deriva dallo sforzo di aiutare lo sviluppo di applicazioni Web.

Il risultato di tutto ciò ha portato in primo luogo ad una evoluzione del modello del markup, che non solo è lievitato per accogliere nuovi elementi, ma modifica in modo sensibile anche le basi della propria sintassi e le regole per la disposizione dei contenuti della pagina, a questo segue un rinvigorimento delle API JavaScript che vengono estese per supportare tutte le funzionalità di cui un'applicazione moderna ha bisogno:

- regole per la strutturazione del testo in capitoli, paragrafi e sezioni;
- introduzione di elementi di controllo per i menu di navigazione;
- introduzioni di elementi specifici per il controllo di contenuti multimediali audio e video;
- introduzione dell'elemento che permette di utilizzare JavaScript per creare animazioni e grafica vettoriale;
- salvare informazioni sul dispositivo client;
- accedere all'applicazione anche in assenza di una connessione di rete;
- comunicare in modo bidirezionale sia con il server sia con altre applicazioni;
- eseguire operazioni in background;
- editare contenuti anche complessi, come documenti di testo;
- pilotare lo storico della navigazione;
- usare tipi di interazioni tipiche di applicazioni desktop, come drag end drop;
- generare grafica in 2D e 3D in tempo reali;

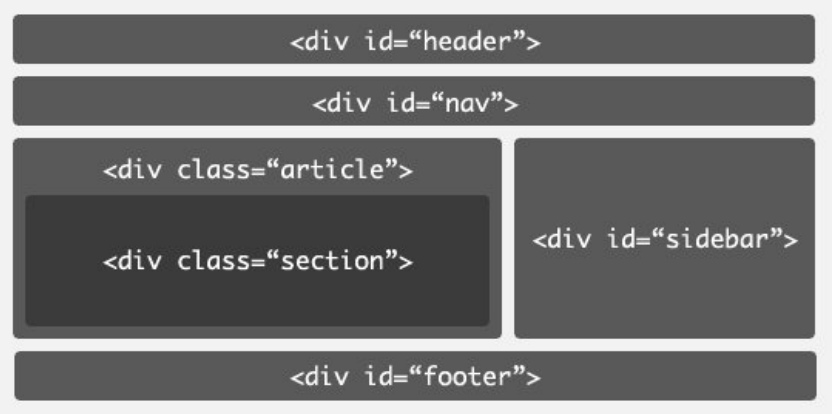

Figura 1.3: Struttura Pagina Html4

- accedere e manipolare informazioni in tempo reale dall'utente attraverso sensori multimediali come microfono e webcam.

Ma non è tutto, attorno al nucleo delle specifiche vi sono tutta una serie di iniziative, alcune delle quali anche in avanzato stato di definizione, studiate per accedere alle informazioni geografiche del dispositivo dell'utente e mantenere un database sul dispositivo utente. Infine senza dimenticare che l'evoluzione dell'HTML viaggia di pari passo con quella dei CSS, che si avvicinano alla terza versione, e di altri importanti standard come SVG (Scalable Vector Graphics) e MathML (Mathematical Markup Language).

1.4.1 Semantics

Alla base della struttura di una pagine in Html4 regnava il tag "div", utilizzatissimo dai web designer, esso consentiva di definire una divisione o una sezione all'interno di un documento Html oppure veniva utilizzato per raggruppare gli elementi in blocchi (footer, header, etc..) formattandoli secondo gli stili. (Vedi Figura 1.1)

Ora Html5 ha completamente eliminato dalla sua vasta gamma di tag il famoso div, e ha introdotto nuovi tag, ciascuno dei quali è andato a sostituire quello che un tempo era definito tramite div. (Vedi Figura 1.2)

Come possiamo notare la nuova struttura l'Html5 aggiunge nuovi elementi distintivi per progettare la pagina:

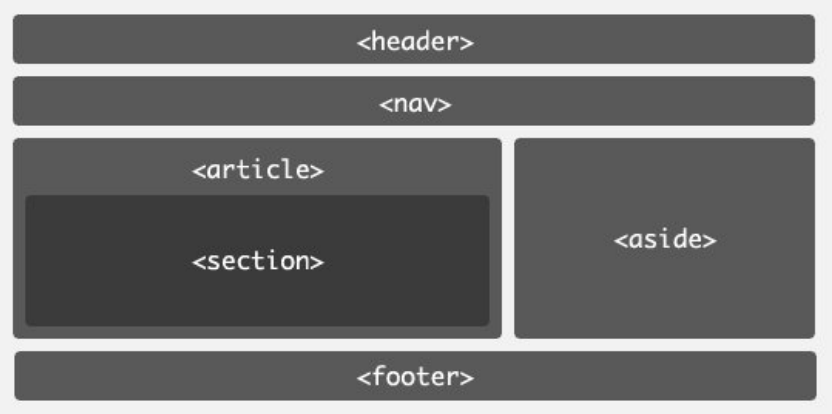

Figura 1.4: Struttura Pagina Html5

- **header**: intestazione visualizzata all'inizio della pagine o di una sezione, infatti potrebbero esserci più tag header all'interno di un documento;
- **nav**: racchiude una serie di link ad altre pagine interne o esterne al sito;
- **section**: corrisponderebbe ad un capitolo di un libro o ad una sezione ben precisa;
- **article**: contiene testo "indipendente", quale potrebbe essere un messaggio scritto in un blog, un articolo ecc;
- **footer**: classico pie di pagina, nel quale inserire l'indirizzo email per i contatti, copyright ecc;
- **aside**: rappresenta una nota, un suggerimento, una barra laterale o qualcosa che si trova solitamente al di fuori del flusso principale di un articolo.

Questa nuova struttura risulta molto utile, anche combinata ai tag H1-H6, in quanto permette la creazione di sezioni nidificate all'interno di ciascun blocco (Vedi Figura 1.3)

Sarà inoltre possibile per l'utente spostarsi rapidamente da una sezione all'altra per navigare più rapidamente tra i contenuti. Nel caso di un blog,

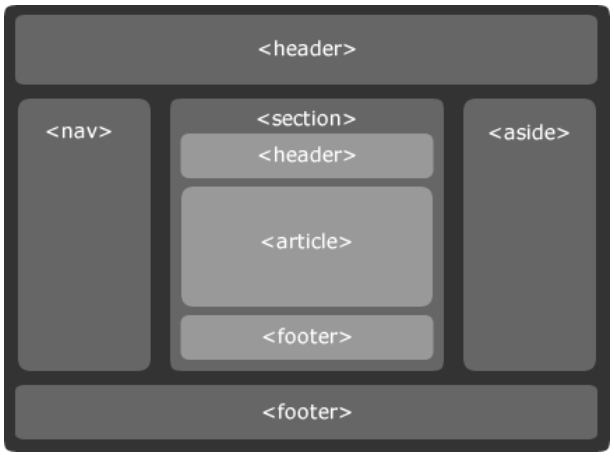

Figura 1.5: Struttura Nidificata Html5

il lettore potrà muoversi dal contenuto di un post a quello successivo senza visualizzare i dati relativi all'autore, alla data di pubblicazione, ecc.

Invece, per quanto riguarda i vantaggi del programmatore, una sorgente pulita da "div" e strutturata in blocchi indipendenti è indubbiamente una cosa da non sottovalutare.

1.4.2 Offline e LocalStorage

Con l'evoluzione del Software as a Service e la crescita esponenziale del numero di applicazioni Web utilizzate, in sostituzione degli ormai "vecchi" software proprietari, è emersa la necessità di introdurre le **Offline Web Application**.

La progettazione di applicazioni Web in grado di funzionare senza un accesso a Internet appare una contraddizione, soprattutto in un'epoca dove tutto ruota attorno al cloud computing e al modello di Software as a Service fornito dal cloud.

Chiaramente, nella realtà dai fatti, è praticamente impossibile che una connessione alla rete Internet sia sempre disponibile e non abbia mai interruzioni. Pertanto per far fronte a tale eventualità, le nuove specifiche Html5 hanno introdotto nuove API per gestire queste caratteristiche.

"Web" e "Online" sono due termini strettamente legati tra loro, addirittura sinonimi per molte persone.

Quindi perché si parla di **tecnologie Web offline**, e che cosa significa tale termine?

Le nuove specifiche Html5 hanno introdotto un certo numero di caratteristiche per rendere le *Web application offline* una realtà:

- application cache;
- localStorage;
- web SQL & indexed database;
- online/offlines events.

Queste caratteristiche possono anche essere utilizzate per migliorare le prestazioni di un'applicazione per l'archiviazione dei dati nella cache e per mantenere i dati delle sessioni in fase di ricaricamento e di ripristino delle pagine.

Per definire il concetto "offline", occorre in primo luogo fare una distinzione fra le diverse tipologie di applicazioni Web.

Si può parlare di applicazioni completamente offline facendo riferimento a tutte quelle applicazioni che vengono eseguite all'interno di un browser, ma che non utilizzano la connessione ad Internet.

Poi vi è una via di mezzo fra le applicazioni online e quelle offline, cioè le applicazioni "online-offline", o anche chiamate ambiguamente "apps non in linea". Queste applicazioni sono destinate alla sincronizzazione con il cloud. Esse sopravvivono a periodi di inattività, intesa come perdita di collegamento alla rete Internet oppure come assenza di linea. Un esempio classico è rappresentato dalla funzione offline di GMail di Google. Oltre allo scopo di sincronizzare i dati con il cloud, vi possono essere anche applicazioni online-offline utilizzate per mantenere intenzionalmente informazioni a livello locale.

Le applicazioni online-offline hanno lo svantaggio di essere caratterizzate da maggiore complessità, perciò sono soggette a maggiori costi di realizzazione e manutenzione. Il supporto "online-offline", perciò, deve essere giustificato. Per fare un esempio, progettare e realizzare un'applicazione che supporti anche la funzionalità offline per essere eseguita dagli utenti

mentre effettuano viaggi aerei, quinci sprovvisti di connessione, non è una giustificazione sufficiente. Però vi sono situazioni la funzionalità di offline è caratteristica è essenziale per l'utilizzo e la diffuzione di determinate applicazioni.

Html5 offre due principali funzionalità per il supporto offline:

- **Application Cache**: prevede il salvataggio del base logica dell'applicazione e dell'interfaccia grafica.

- **Offline Storage** (o anche "Local Storage"): effettua la cattura di specifici dati generati dall'utente oppure di risorse per cui l'utente ha mostrato interessa.

Per meglio comprendere l'utilizzo di queste funzionalità, prendiamo come esempio un'applicazione Web di un gioco. La funzionalità *application caching* permetterebbe di mantenere in locale l'iniziale documento Html, i contenuti JavaScript e CSS, le icone e immagini utilizzate più di frequente. Di conseguenza, alla successiva visita dell'utente, il caricamento della pagina sarebbe immediato.

I browser utilizzati fin ad oggi, hanno sempre utilizzato una propria cache per la memorizzazione. Qual'è la novità che differenzia la cache classica dall'application cache? La classica cache del browser ha sempre permesso di memorizzare i contenuti delle pagine web, ma tale cache è facilmente sovrascrivibile. L'application cache, a differenza, è destinata ad avere un proprio "status" speciale all'interno del disco rigido dell'utente, in modo tale che non possa essere spostata o sovrascritta perché è stato scaricato un video di grandi dimensioni.

La funzionalità di application cache è stata aggiunta per affiancare la cache classica e non per sostituirla. Per questo, sono stati aggiunte delle API per effettuare il controllo di ciò che è memorizzato nella "nuova" cache e ciò che non lo è.

Grazie all'application cache visitando successivamente l'applicazione del gioco i caricamenti saranno immediati, ma come ci si dovrebbe comportare se si volesse consentire all'utente di continuare a giocare a partire dall'ultimo "salvataggio", cioè com'è possibile rispristina lo stato di avanzamento del gioco?

I dati che riguardano lo stato di avanzamento del gioco, non sono tipi di dati memorizzati all'interno della cache application, perché sono dati specifici dell'utente e non dati generali riguardanti l'applicazione del gioco.

Questi dati perciò non riguardano più l'application cache, ma riguardano l'*offline storage*.

Ogni volta che l'utente effettua il salvataggio dei progressi del gioco, viene effettuato il caricamento sul cloud. Oltre al caricamento online, allo stesso tempo viene effettuata la memorizzazione dei dati sul disco rigido dell'utente, cioè viene effettuata una conservazione non in linea dei dati del gioco. Ad esempio, i dati del salvataggio offline potrebbero riguardare il livello di vita del giocatore e lo stato di avanzamente o del personaggio all'interno del mondo del gioco.

Possono essere applicate diverse strategie di storage offline, dipende dalle scelte di progettazione che si vogliono effettuare. Ad esempio, si potrebbero mettere in atto cicli continui di memorizzazione offline dei dati del gioco, in questo modo anche le ultime modifiche di stato del gioco verranno salvate, in caso di perdita della connessione oppure in caso di chiusura anticipata del browser prima del salvataggio.

Alternativamente si può effettuare la scelta di utilizzare meno la larghezza della banda di connessione, mettendo in atto, per esempio, il salvataggio dei dati in locale ogni 10 secondi, mentre l'upload dei dati sul cloud ogni 5 minuti. In questo modo si ha il risparmio di banda.

Riassumendo l'application cache rappresenta uno strumento tramite il quale è possibile specificare al browser quali file salvare, non nella cache "classica" del browser ma in una cache apposita, definita per l'appunto "application cache", così da permettere una navigazione il più possibile completa, anche quando la connessione non è disponibile e si è offline.

Il vantaggio principale dell'application cache rispetto a quella tradizionale è rappresentato dal controllo. Nella cache classica è il browser a decidere quali sono i file da tenere in memoria, invece, con lo strumento offerto da Html5 è possibile specificare con precisione quali sono le risorse da memorizzare.

L'application cache consente anche di salvare gli script utilizzati nell'applicazione, in modo tale da consentire un utilizzo totale dell'applicazione anche offline, non una semplice e solo consultazione.

Ulteriore vantaggio, infine, è rappresentato dalla possibilità da parte dell'application cache di poter salvare anche file che non si sono visitati, ma che potrebbero essere utili per una navigazione completa, mentre nella cache classica del browser una pagina per essere memorizzata deve essere stata visualizzata almeno una volta.

Affianco al concetto di application cache, vi è quello di offline storage. Esso rispetto ai cookies, rappresenta la soluzione ideale per progetti in cui si ha la necessità di salvare i dati e lo "stato" di un'applicazione. L'unico svantaggio che caratterizza questa funzionalità è che non vi è modo di specificare la "scadenza" dei dati che vengono memorizzati, cosa che invece i cookies concepiscono. La durata dei dati memorizzati è una procedura che deve essere gestita autonomamente.

Le nuove API introdotte da Html5 per l'utilizzo di queste funzionalità sono:

- Web Storage (chiamato anche "Local Storage" o "DOM Storage");
- Web SQL (o semplicemente "SQL Storage");
- IndexedDB;
- File Storage

Html5, offrendo tali funzionalità, permette di progettare e realizzare applicazioni in grado di supportare tutte quelle caratteristiche che in precedenza venivano realizzate con plugins esterni e API specifiche.

L'application cache, combinato con il local storage, sono strumenti molto potenti che consentono di creare delle vere e proprie applicazioni indipendenti dall'effettiva connessione ad Internet o meno.

L'offline storage e l'application cache hanno molte potenziali applicazioni, che vanno ben oltre l'ovvio scenario "Voglio che la mia applicazione possa essere eseguibile quando gli utenti sono in aereo".

Gli sviluppatori in passato hanno dovuto ricorrere a soluzioni alternative, come API specifiche e plugins dei browser, ma gli sviluppatori attuali hanno a disposizione una serie di funzionalità che fino ad oggi non era possibile sfruttare in tutta la funzionalità. L'unico dilemma sta nel scegliere quale usare e soprattutto come sfruttarle nel modo giusto, in modo tale da creare applicazioni performanti, a livello delle passate applicazioni proprietarie.

1.4.3 Multimedia e Grafica

Da quando il Web ha smesso di essere un mezzo per la consultazione degli ipertesti, cioè dal passaggio dal Web 1.0 al 2.0, è sempre stato un mezzo di navigazione visivo, anche se con delle restrizioni.

Fino a poco tempo fa, gli sviluppatori si sono limitati all'uso di Html, Css e JavaScript per produrre animazioni ed effetti grafici all'interno dei siti Web. Gli sviluppatori se desideravano creare giochi, animazioni o sofisticati effetti visivi erano costretti a rivolgersi a piattaforme diverse oppure utilizzare i plugin.

Con l'introduzione delle specifiche Html5, il browser è diventato una vera e propria piattaforma per i giochi, animazioni, film e qualsiasi cosa grafica. Grazie all'introduzione di nuove tecnologie, da parte di Html5, come *canvas*, *immagini SVG*, *audio* e *video*, ora è possibile realizzare applicazioni ad alta prestazione grafica come 3D, grafica vettoriale, animazioni etc, senza l'usi di strumenti esterni a Html o JavaScript.

Il nuovo tag canvas permette il rendering dinamico delle immagini bitmap gestibili attraverso un linguaggio di scripting, come per esempio JavaScript. Fu inizialmente introdotto da Apple per migliorare applicazioni come Dashboard widgets ed il browser Safari.

Canvas consiste in una regione disegnabile, definita in codice html tramite gli attributi "height" e "width". Il codice JavaScript ha la possibilità di accedere all'area con un set completo di funzioni per il disegno, simili a quelle comuni ad altre API 2D, permettendo così la generazione dinamica di disegni. Alcuni usi possibili di canvas includono i grafici, l'animazione e la composizione di immagini.

SVG, altra novità introdotta da Html5, a differenza di canvas, permette la realizzazione di immagini in 2D in XML.

Ogni elemento SVG, basandosi su XML, è disponibile all'interno del DOM della pagina, perciò è possibile registrare tu tali elementi degli event handler JavaScript, per la gestione degli eventi.

In SVG, ogni forma disegnata è identificata come un oggetto. Perciò se gli attributi di tale oggetto vengono modificati, il browser automaticamente aggiorna la forma disegnata.

Date le particolari differenze emerse tra i componenti canvas e SVG, quando si progetta la grafica per una applicazione, in base alle caratteristiche e all'uso, occorre decidere l'uso di una o dell'altra tecnologia.

Html5 oltre ad offrire API per la grafica, ha messo a disposizione un set di API, ancora oggi in costruzione, per la grafica in 3D. Queste API fatto parte di un libreria di istruzioni denominate WebGL (Web-based Graphics Library).

Oltre ai tag di tipo grafico, non bisogna scordare i nuovi elementi multimediali introdotti da Html5: audio e video.

I nuovi tag audio e video permettono di inserire in maniera facile, veloce e senza l'utilizzo di plugins esterni, come per esempio Flash e SilverLight, file audio e file video. Ciò che emerge da tutto ciò è che, contrariamente a quanto avviene ad esempio in Flash, questi nuovi elementi sono parte del DOM della pagina e dunque possono essere liberamente modificati e gestiti dinamicamente da JavaScript, ad esempio in combinazione con altri oggetti Html5 come canvas, per la manipolazione in tempo reale di un video.

Naturalmente, nessuna di queste nuove tecnologie sarebbero utili se il browser non fosse in grado di eseguire il tutto velocemente. Fortunatamente, i "motori" JavaScript, con l'evoluzione del linguaggio, sono diventati abbastanza performanti da eseguire, per esempio, giochi in 3D e manipolare i video in tempo reale.

Gli sviluppatori stanno usando tali caratteristiche per creare applicazioni, con interfacce grafiche caratterizzate da elementi che non hanno vincoli di utilizzo di API esterne o plugins proprietari.

1.4.4 WebWorker

Le nuove specifiche Html5 hanno portato notevoli migliorie non solo dal punto di vista del markup semantico o della possibilità di personalizzare l'aspetto grafico della pagina. Sono state introdotte anche notevoli migliorie e implementazioni dal punto di vista di JavaScript. I browser moderni, infatti, dovranno supportare una serie di API con diversi scopi che faciliteranno il compito degli sviluppatori e permetteranno di avere applicazioni web ancora più performanti e vicine come usabilità a quelle desktop.

Una delle novità più interessanti è rappresentata dai WebWorkers. Grazie ad essi è possibile introdurre nell'ottica delle Web application la possibilità di supportare il concetto di concorrenza che fino ad ora non era realizzabile.

Ciascun WebWorker rappresenta un entità che consente di eseguire codice JavaScript in un thread separato dal thread principale utilizzato per la

User Interface. Ognuno di essi, viene identificato da un particolare file che implementa ciò che il WebWorker deve eseguire. Tali sotto-script vengono eseguiti in uno scope particolare, all'interno del quale non sarà possibile accedere agli elementi DOM della pagina, proprio per il fatto che essi devono agire indipendentemente dalla pagina e sono predisposti per eseguire operazioni complesse senza però bloccare la User Interface, come invece potrebbe accadere in architetture che non sfruttano questa potenzialità.

I WebWorker possono comunicare con l'ambiente esterno mediante lo scambio di messaggi. Lo scambio di messaggi avviene tramite l'utilizzo di una struttura orientata agli eventi e in particolare grazie al metodo "postMessage" è possibile inviare messaggi. Questi messaggi sono gestiti in modalità asincrona grazie alla proprietà "onmessage" associato al WebWorker.

La limitazione che caratterizza i WebWorker consiste nell'impossibilità in alcun modo di poter accedere direttamente al DOM della pagina, cioè non è possibile modificare il markup della pagina tramite un WebWorker. Ciò non significa che non è possibile interagire con la pagina principale tramite un WebWorker, ma è necessario rispettare degli appositi canali di comunicazione, nel nostro caso i canali di comunicazione fra pagina e WebWorker sono rappresentati dagli eventi.

Oltre a tutto ciò, i WebWorker non solo possono eseguire porzioni di codice in un contesto separato rispetto alla pagina principale, ma hanno la possibilità di creare e di utilizzare altri WebWorker e comunicare con essi con le stesse API utilizzate in precedenza per far comunicare la pagina principale con la WebWorker. Questa tecnica di delegazione da un lato permette di ottimizzare ancora di più i nostri script sfruttando al meglio i processori multicore ma dall'altro presenta notevoli difficoltà dal punto di vista dello sviluppo e dell'integrazione tra i diversi componenti.

1.4.5 WebSocket

Sin dalle sue origini, il Web è stato in gran parte costruito attorno al cosiddetto paradigma richiesta/risposta di HTTP.

L'architettura del Web è sempre stata caratterizzata da una struttura Client-Server, dove nella modalità di comunicazione vi è un Client ben definito che invia una richiesta al Server, ed esso fornisce una risposta a ciò che è stato richiesto dal Client. Se questa struttura risulta essere perfetta per

un Web di tipo statico, fatto di pagine collegate fra loro, in cui la richiesta di un nuovo contenuto è generata dal click su un link da parte del Client, questa struttura non è per nulla ideale per Web application complesse dove sono necessarie comunicazioni di tipo bidirezionali o dove le informazioni potrebbero dover arrivare direttamente dal Server, senza nessuna precedente richiesta da parte del Client.

Tecnologie che consentono al Server di inviare dati ad un Client, nel momento stesso in cui si sa che i dati sono disponibili, possono essere ad esempio "Comet" oppure "Long Polling". Tuttavia tutte queste tecnologie risolvono la questione della comunicazione da parte del Server condividendo un problema di base: tutti portano ad un sovraccarico dell'HTTP, che non li rende adatti per applicazioni a bassa latenza. Basti pensare ai primi giochi Web multiplayer o qualsiasi altro gioco online caratterizzati da componenti realtime.

Html5 ha tentato di risolvere tali problematiche introducendo la specifica WebSockets.

La specifica WebSockets è costituita da un insieme di API che permettono di stabilire una connessione tra un Client browser e un Server. In altre parole, permettono di creare una connessione persistente tra il Client e il Server, ed entrambe le parti possono iniziare ad inviare dati in qualsiasi momento, quello che si viene a creare è una comunicazione bidirezionale fra Client e Server.

Per aprire una nuova connessione WebSockets occorre semplicemente creare un nuovo oggetto WebSocket.

```
var connection = new WebSocket('ws://theSite.com/echo',
                                        ['soap','xmpp']);
```

È possibile notare nel primo argomento l'utilizzo di "ws:". Questo rappresenta il nuovo schema URL per le connessioni WebSockets. È anche possibile utilizzare "wss:" per il collegamento a WebSocket sicuri, un po' come per HTTP viene utilizzato "https:" per le connessioni HTTP sicure.

Registrando immediatamente alcuni event handler sugli eventi, come per esempio "onopen", "onerror" o "onmessage", della connessione WebSocket effettuata, è possibile permettere di conoscere quando la connessione è aperta, quando vi è un messaggio in arrivo oppure quando accade un errore. Tramite la registrazione degli eventi sulla connessione socket è possibile tenere sotto controllo lo stato della connessione.

Il secondo argomento utilizzato nella procedura di creazione della connessione socket, rappresenta i "sottoprotocolli" opzionali. Tale argomento può essere una stringa o un array di stringhe, ognuna delle quali rappresenta il nome di un "sottoprotocollo". Il server è in grado di accettare un solo tipo di protocollo facente parte dell'array. Determinare quale "sottoprotocollo" può essere utilizzato dal Server è possibile mediante il primo accesso alla proprietà dell'oggetto WebSockets.

I nomi dei "sottoprotocolli" devono essere necessariamente registrati nel registro IANA. Attualmente vi è un solo nome di "sottoprotocollo", cioè il "soap", registrato a partire da Febbraio 2012.

Non appena la connessione al server è stata stabilita, è possibili iniziare ad inviare dati al server utilizzando il metodo "send" sull'oggetto connessione creato in precedenza.

```
connection.send('Hello Server!!');
```

Allo stesso modo il Server può inviare messaggi al Client in qualsiasi momento. Ogni volta che ciò accade, viene lanciato la funzione di callback registrata sull'evento "onmessage". Tale callback riceve l'event object ed il messaggio effettivo inviatogli, accessibile tramite la proprietà "data".

Le specifiche WebSockets sono una tecnologia giovane, non pienamente supportata da tutti i browser. Tuttavia, è possibile ad oggi utilizzare le WebSockets mediante librerie che utilizzano le modalità descritte qui sopra, questo quando le WebSockets non sono disponibili.

Le WebSockets sono una tecnologia utilizzata ogni volta che si ha la necessità di applicazioni caratterizzate da una latenza molto bassa, molto vicina alla connessione realtime tra Client e Server. Alcuni esempi di utilizzo possono essere:

- applicazioni chat in realtime;
- giochi online multiplayers;
- aggiornamento in tempo reale di informazioni.

Capitolo 2

JavaScript

2.1 Cos'è JavaScript

Contrariamente a quanto ci suggerisce il nome, JavaScript ha ben poco a che fare con il linguaggio di programmazione chiamato Java.

Il nome simile è stato ispirato più per ragioni di marketing piuttosto che per l'utilizzo e le funzionalità che esso svolge.

JavaScript è un linguaggio di scripting lato-client, ma soprattutto è il linguaggio di scripting per eccellenza e certamente quello più usato.

Fra le diverse tipologie di linguaggi, compilati, semi-compilati e interpretati JavaScript fa parte di quella categoria di linguaggi che vengono interpretati, nel nostro caso dal browser.

Un linguaggio di scripting, infatti, è un linguaggio di programmazione interpretato, cioè il codice viene proposto in "chiaro", senza nessuna precedente compilazione come ad esempio nei linguaggi compilati e semi-compilati come C e Java. L'operazione di interprete spetta ad un programma che deve interpretarlo ed eseguirne le istruzioni contenute.

Nei linguaggi compilati e semi-compilati è possibile scrivere programmi con una determinata sintassi, la quale poi viene passata ad un compilatore che trasforma un determinato linguaggio di programmazione (codice sorgente) in linguaggio macchina (codice oggetto).

A differenza di altri linguaggi di programmazione, come ad esempio Java (con il quale non è assolutamente da confondere), che permette la scrittura

di programmi completamente stand-alone, JavaScript viene utilizzato soprattutto in quanto linguaggio di scripting, integrato, quindi, all'interno di un altro programma.

L'idea di base consiste nell'avere un "programma ospite", cioè colui che ospita ed esegue lo script, che fornisce un'API ben definita. Lo script, quando eseguito, utilizza riferimenti a questa API per richiedere, al "programma ospite", l'esecuzione di operazioni specifiche, non previste dai costrutti del linguaggio JavaScript in sé.

L'esempio tipico (e, forse, il più noto) di "programma ospite" per uno script Javascript è quello del browser, programma che permettere di leggere pagine HTML.

Ogni browser tipicamente incorpora un apposito motore di scripting (cioè di visualizzazione) che legge le parti del codice JavaScript e le esegue nel momento in cui quest'ultimo è presente all'interno di una pagina web.

L'uso principale del Javascript in ambito Web è la scrittura di piccole porzioni di codice integrate all'interno delle pagine HTML che interagiscono con il DOM del browser per compiere determinate azioni che non sarebbero possibili con il solo HTML statico. Infatti la tecnologia Javascript viene impiegata lato client per creare vere e proprie potenti applicazioni web dinamiche.

2.2 Un po' di storia

JavaScript fu originariamente sviluppato nel 1995 da Brendan Eich, collaboratore della Netscape Communications, con il nome di LiveScript (chiamato così per la sua vivacità e dinamicità).

Netscape voleva dotare il proprio browser di un linguaggio di scripting che permettesse ai web designer di interagire con i diversi componenti della pagina (immagini, form, link, ecc.), ma soprattutto con le applet Java (programmi che permettono l'interazione con l'utente).

In quel periodo Netscape era particolarmente vicina alla Sun Microsystems (ideatrice di Java), con cui aveva stretto una partnership.

Le due aziende, alla fine del 1995, annunciarono la nascita della prima versione di questo nuovo linguaggio, descrivendolo come "complementare all'HTML e a Java", ribatezzandolo JavaScript, per motivi di marketing data

l'assonanza con il linguaggio Java, che rappresentava una delle tecnologie più avanzate per l'epoca, con cui JavaScript non aveva niente in comune, se non per la sintassi.

La versione 2.0 del browser Netscape Navigator, che incorporava JavaScript, fu un grande successo, ma i webdesigner non utilizzarono JavaScript per interagire con le applet Java (come avrebbe voluto Netscape), ma preferirono utilizzarlo per rendere dinamiche le pagine.

Dopo il decollo e il successo che JavaScript aveva acquisito, Microsoft decise di aggiungere al proprio browser Internet Explorer un proprio linguaggio di scripting, JScript.

JScript aveva notevoli differenze con la versione di Javascript sviluppata da Netscape, per questo motivo nacque l'esigenza di creare un standard.

Il processo di standardizzazione fu affidato all'European Computer Manufacturers Association (ECMA), la stessa associazione che oggi è incaricata da Microsoft di standardizzare il C#.

ECMA realizzò lo standard del linguaggio di scripting e lo battezzò ECMAScript.

ECMAScript è dunque il "figlio" di JavaScript, e ad oggi quando si parla di JavaScript, JScript o ECMAScript sostanzialmente si indicano tre varietà dello stesso linguaggio.

Ad oggi siamo alla quinta versione di ECMAScript (ECMAScript5) la quale verrà poi sottoposta all'ISO.

2.3 Perché usare JavaScript

Oggigiorno in ogni sito Web che si rispetti c'è sempre un pó di JavaScript: anche un semplice effetto "roll-over", magari ottenuto con qualche software grafico, nasconde tra le righe del codice HTML un pó di sintassi JavaScript. Infatti, con questo semplice linguaggio di scripting che viene interpretato dal browser, si possono eseguire un'infinità di operazioni che il semplice HTML non permette.

JavaScript nonostante sia un linguaggio di programmazione leggero, incorporato direttamente all'interno delle pagine HTML, senza l'acquisto di una licenza per l'utilizzo:

- offre hai progettisti uno strumento di programmazione HTML con una sintassi di scrittura molto semplice;

- è in grado di reagire agli eventi, come per esempio per l'uso del mouse o della tastiera etc..;
- è in grado di leggere e modificare il contenuto di un elemento HTML;
- può essere utilizzato per convalidare i dati delle form prima che vengano inviati a un server. Ciò consente di risparmiare al server l'elaborazione aggiuntiva per i dati inviati.
- può essere utilizzato per individuare il browser dell'utente e, a seconda del browser, caricare un'altra eventuale pagina specificatamente progettata per quel browser;
- può essere utilizzato per creare i cookie, cioè archiviazione e recupero di informazioni sul computer dell'utente.

2.4 Programmazione Object-Oriented

JavaScript oltre a differire da JavaScript per il nome, differisce anche per la tipologia di linguaggio: JavaScript è un linguaggio di programmazione **Object-Based** e non Class-Based, come Java. Entrambe le tipologiasi basano sull'utilizzo degli oggetti, ma fondamentalmente sono due cose differenti.

Un **oggetto** è un'entità composta da proprietà e metodi. Le proprietà rappresentano i "dati" che compongono l'oggetto, invece i metodi rappresentano le specifiche per la manipolazione delle proprietà dell'oggetto.

I linguaggi di programmazione Class-Based sono linguaggi di programmazione orientati agli oggetti basati sul concetto di **classe**.

Un oggetto deve essere esplicitamente creato sulla base di una classe e un oggetto così creato, viene considerato una "istanza" di quella classe. Questo perché, la classe definisce la struttura e il comportamento che un oggetto deve assumere.

La pura conseguenza del concetto di classe è l'**ereditarietà**.

L'ereditarietà rappresenta il legame che si viene a stabilire fra le diverse classi. Questo legame stabilisce la presenza di classi che hanno la possibilità di ereditare/acquisire le proprietà e i metodi da classi pre-esistenti denominate classi base o superclassi. Le nuove classi sono conosciute come classi derivate o sottoclassi.

Una classe dichiarata sottoclasse di una classe base:

- eredita tutte le variabili e tutti i metodi della classe base;
- può avere proprietà o metodi aggiuntivi;
- può ridefinire i metodi ereditati attraverso l'overriding, in modo che essi eseguano la stessa operazione in un modo più specializzato.

I linguaggi di programmazione Object-Based, come JavaScript, sono linguaggi di programmazione che si basano sugli oggetti, senza prevedere l'utilizzo del concetto di classe.

In questa tipologia, gli oggetti sono visti come pure strutture dati, contenenti proprietà e metodi per la manipolazione. Non prevedendo l'uso delle classi viene a "cadere" il concetto di ereditarietà, definito per i linguaggi Class-Based, nonostante esso abbia importantissime ripercussioni sulla riusabilità del software e sia un importante strumento di modellazione.

Per compensare questa "carenza", i linguaggi Object-Based utilizzano lo stile di programmazione Prototype-Based. Lo stile di programmazione Prototype-Based, non avendo a disposizione le classi, per "simulare" il concetto di ereditarietà utilizza un processo di "clonazione" degli oggetti già esistenti.

2.5 JavaScript e DOM

Il linguaggio JavaScript come interagisce con l'html e con il browser?

Ogni browser, con all'interno il documento html, può essere immaginato come un insieme di elementi messi in relazione tra loro.

Infatti, il browser, può essere "sezionato" nei seguenti principali elementi:

- prima di ogni altra cosa c'è il browser stesso (oggetto "Navigator");
- la finestra che contiene tutti gli elementi (oggetto "window");
- il documento Html vero e proprio (oggetto "document);
  - form per gli imput dell'utente (oggetto "document.forms");
  - immagini (oggetto "document.images");

  - cookie (oggetto “document.cookie”);

- la barre degli indirizzi (oggetto “location”);

Le regole che mettono in relazione gli elementi che fanno parte di un browser sono gestiti dal DOM.

Il DOM (Document Object Model) è un'interfaccia di programmazione (API) per documenti Html, che consente a Javascript di rapportasi con un browser e con gli elementi contenuti in esso.

Il DOM (Document Object Model), letteralmente modello ad oggetti del documento, definisce la struttura logica dei documento e degli elementi che ne fanno parte, e specifica il modo in cui questi documenti sono accessibili e modificabili.

Con il DOM (Document Object Model), gli sviluppatori sono in grado di creare documenti, navigarne la loro struttura, e di aggiornare, modficare e eliminare elementi e contenuti. Nativamente supportato dai browser per accedere e modificare gli elementi di un documento Html, DOM è un modo per accedere e aggiornare dinamicamente il contenuto, la struttura e lo stile di un documento.

Nel corso degli anni, date le numerose incompatibilità dovute al diverso tipo di gestione del DOM da parte dei vari browser, il W3C (World Wide Web Consortium) ha stabilito delle specifiche DOM standard.

Le specifiche DOM elaborate da W3C sono suddivise in livelli, ciascuno dei quali contiene moduli obbligatori e opzionali.

- **Livello 0**
- **Livello 1**
- **Livello 2**
- **Livello 3**

Le specifiche DOM del W3C hanno come obiettivo quello di fornire un'interfaccia standard di programmazione che può essere utilizzabile in un'ampia varietà di ambienti e applicazioni.

Il DOM è un progetto che può essere utilizzato con un qualsiasi linguaggio di programmazione. JavaScript e l'interazione con i documenti Html è uno dei casi di utilizzo delle specifiche DOM.

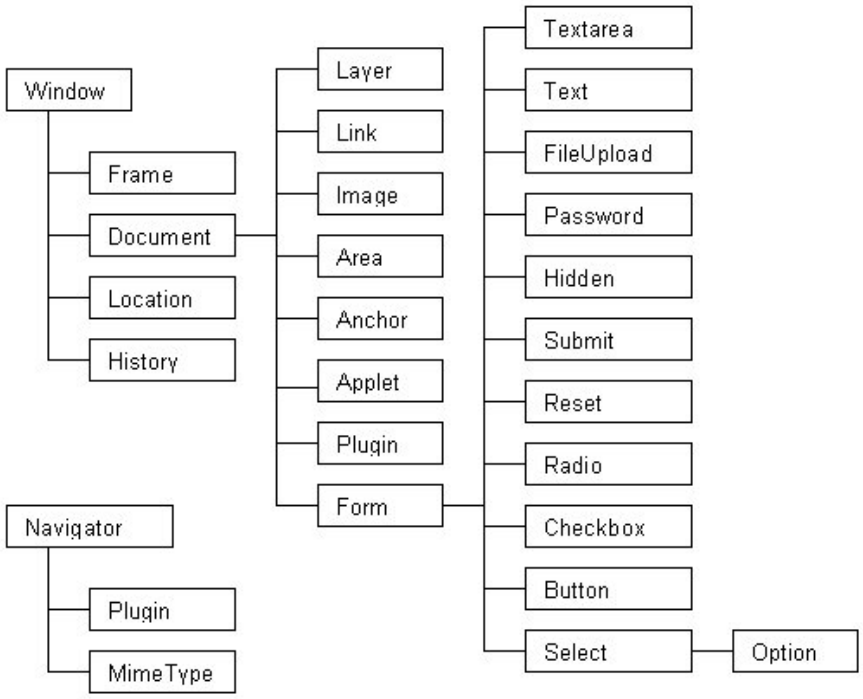

Figura 2.1: Struttura DOM Browser

Ad oggi il DOM definito dal W3C, è indipendente dal tipo di browser, di versione e di sistema operativo.

Un browser per sostenere di appartenere ad un certo "livello" dello standard W3C, deve soddisfare tutti i requisiti del livello a cui vuole appartenere e di tutti i livelli inferiori.

La specifica attuale di DOM è al "Livello 2", tuttavia il W3C sta già lavorando sulle nuove specifiche del "Livello 3".

Nel DOM (Document Object Model), i documenti hanno una struttura logica molto simile ad un "albero", o per essere più precisi, ad un "foresta", perché un documento può contenere più di un "albero". (Vedi Figura 2.1)

Tuttavia il DOM non specifica se un documento possiede una struttura ad "albero" o a "foresta", ma specifica solo quali sono i rapporti fra gli elementi contenuti nel documento.

Per evitare la terminologia "albero" o "foresta", d'ora in poi verrà

utilizzato il termine "modello di struttura".

Una proprietà importante dei modelli di struttura DOM è l'**isomorfismo strutturale**: se due implementazioni del DOM sono utilizzate per creare una rappresentazione dello stesso documento, essi creeranno la stessa struttura di modello, con esattamente gli stessi oggetti e le stesse relazioni.

Il nome "Document Object Model" è stato scelto perché è un "modello ad oggetti", cioè nell'ottica della tradizionale progettazione orientata agli oggetti: i documenti sono modellati utilizzando oggetti, e il modello non comprende solo la struttura del documenti, ma anche il comportamento che un documento e gli oggetti che lo compongono devono avere.

In altre parole, i nodi del diagramma ad "albero", non rappresentano una struttura di dati, ma rappresentano oggetti con identità e funzionalità.

Essendo un modelli ad oggetti, il DOM identifica:

- le interfacce e gli oggetti usati per rappresentare e manipolare un documento;
- la semantica di queste interfacce e oggetti, tra cui comportamento e attributi;
- le relazioni presenti fra tali interfacce e oggetti.

Nei modelli di dati astratti, il tutto è incentrato tutto intorno ai dati. Invece nel modello ad oggetti, i dati vengono incapsulati all'interno degli oggetti, i quali nascondo i dati, proteggendoli dalla manipolazione diretta dall'esterno.

Le funzioni associate a questi oggetti, determinano come gli oggetti possono essere manipolati.

2.6 La Struttura di JavaScript

2.6.1 Valori, Variabili

All'interno del mondo del computer, ci sono solo dati. Tutto ciò che non è un dato, non esiste. Anche se in sostanza tutti i dati sono una sequenza di bits, e quindi fondamentalmente sono la stessa cosa, ogni dato ha un ruolo ben definito che deve svolgere, che può differire dal ruolo di un'altro dato.

In JavaScript, la maggior parte dei dati si distinguono in base al cosiddetto valore (del dato). Ogni valore assume un determinato tipo, che determina qual'è il ruolo che il dato può svolgere. Per creare un valore, occorre semplicemente invocarne il nome. Naturalemente questi valori non vengono creati dal nulla, ogni valore deve essere memorizzato da qualche parte in memoria, e li vi rimane finché il valore non è più utilizzabile o di interresse.

Nel nostro caso sono presenti i seguenti tipi primitivi di valori:

- **numeri**: Il valore di tipo numerico, come si deduce, non è altro che un valore numerico.

  ```
  144
  ```

  Non viene fatta una distinzione fra valori numerici interi e reali, invece i numeri frazionari, come float o double, sono scritti utilizzando la virgola.

  ```
  9.81
  ```

  Per i numeri molto grandi o molto piccoli, è possibile utilizzare la notazione scientifica, consiste nell'aggiungere una "e" seguita dall'esponente del numero.

  ```
  2.998e8
  ```

  I calcoli con i numeri interi (anche chiamati Integer) garantiscono la precisione, al contrario dei calcoli con i numeri frazionari che generalmente non lo sono, perché entrano in gioco le approssimazioni. La cosa importante è essere consapevoli di tali approssimazioni.

  Quello che è possibile fare con i valori di tipo numerico sono le operazioni aritmetiche. Le operazioni aritmetiche come l'addizione e la moltiplicazione prendono i due valori numerici e ne creano un nuovo valore dato dal risultato dell'operazione.

  ```
  100 + 4 * 11
  ```

- **stringhe**: Non è facile capire esattamente cosa sono le stringhe in JavaScript.

  Sarebbe possibile dire che mentre in Java sono oggetti che sembrano dati di tipo primitivo, in JavaScript sono dati di tipo primitivo che sembrano oggetti.

  Le stringhe vengono utilizzate per rappresentare del testo, il nome presumibilmente deriva dal fatto che una stringa rappresenta l'insieme di un gruppo di caratteri.

  Le stringhe sono scritte in modo tale da racchiudere il loro contenuto all'interno delle singole o doppie apici.

  ```
  "This is a String"
  ```

  Quasi tutto può essere messo tra apici, ma alcuni caratteri, come per esempio Invio, possono essere ingannevoli.

  Le stringhe non possono essere divise, moltiplicate o sottratte, però, ad esse può essere applicato l'operazione di addizione per effettuare l'operazione di concatenazione fra stringhe.

  ```
  "con" + "cat" + "e" + "nate"
  ```

- **boolean**: infine, ci sono i valori di tipo boolean, che comprende solo due valori possibile "false" e "true"

La creazione di valori di tipo primitivo, anche se è la parte essenziale di ogni programma JavaScript, non è sufficiente.

Come fa un programma a mantenere un proprio stato interno dei valori? Occorre uno strumento chiamato **variabile**.

Le variabili sono dei nomi simbolici che servono ad individuare delle locazioni di memoria in cui possono essere posti dei valori.
In JavaScript le variabili vengono create con la dichiarazione "var".

```
var result = ;
```

Per le variabili che si dichiarano e si inizializzano senza darle un particolare valore si attribuisce il valore "null". Questo valore può apparire inutile, ma diventa essenziale se si vuole verificare la presenza in memoria di una variabile.

```
var result = null;
```

Un'altro valore che viene assegnato alle variabili quanto queste sono state dichiarate ma senza assegnamento è "undefined'. Questo valore è stato introdotto con JavaScript 1.3 e accettato dall'ECMAScript.

```
var result;
```

L'insieme di variabili e dei loro valori che esistono in un determinato momento viene chiamato "environment".

Quando un programma si avvia, questo "environment" non è vuoto, contiene sempre un certo numero di variabili standard.

Il browser avviandosi, crea un nuovo "environment" contenente le variabili standard e tutte le variabili create e modificate all'interno del programma. Queste ultime esistono finché il browser non esegue un cambio di pagina.

Quando si parla di variabili occorre fare distinzione fra **variabili globali** e **variabili locali**, la distinzione nasce a seconda di dove vengono dichiarate la variabili.

La distinzione tra variabili globali e variabili locali non è cosa da niente, anzi è alla base della programmazione orientata agli oggetti:

- le variabili globali hanno valore per tutto il documento HTML e vanno dichiarate all'inizio dello script e fuori da ogni funzione.

- le variabili locali hanno valore solo all'interno della funzione in cui sono dichiarate.

I due differenti tipi sono generati da esigenze diverse, in quanto le variabili locali hanno una vita brevissima ed esistono finché opera la funzione, all'uscita della funzione vengono distrutte liberando la memoria, mentre le variabili globali sono dei "contenitori" che durano per tutta la durata della pagina e servono a veicolare dei valori tra le funzioni e tra gli script, ma anche tra le varie pagine o al sever. L'utilizzo delle variabili locali sta nella modularità del linguaggio; le variabili locali rendono uno script riutilizzabile anche in altre pagine HTML, soprattutto se salvato in un file esterno con estensione js.

2.6.2 Controllo di Flusso

Subito dopo la definizione di valore e di variabile, non può altro che esserci, come definizione, quella di controllo di flusso.

Un controllo di flusso è un controllo che viene eseguito durante il flusso di esecuzione di un programma, esso è caratterizzato da un'istruzione condizionata, un'espressione che controlla il verificarsi o meno di una o più determinate condizioni.

I più utilizzati controlli di flusso sono:

- **If...Else**

  ```
  if (<espressione>)
  <istruzione>|{<blocco di istruzioni>}
  else
  <istruzione>|{<blocco di istruzioni>}
  ```

  Tale controllo esegue il primo blocco di istruzioni se l'espressione condizionata da esito "true", in alternativa, se da esito "false", viene eseguito il secondo ramo di istruzioni.

- **For**:

  ```
  for(<inizializzazione>; <condizione di fine ciclo>; <incremento>)
  <istruzione>|{<blocco di istruzioni>}
  ```

  Il For è il più classico tra i costrutti per le iterazioni. Esso esegue ciclicamente le istruzioni al suo interno finché non viene raggiunto il limite indicato dalla condizione di fine ciclo.

  La condizione di fine ciclo viene verificata tipicamente su una variabile utilizzata come contatore. Un ciclo si compone delle seguenti tre operazioni:

  – inizializzazione della variabile, in cui viene definito il valore di partenza;

  – la verifica della condizione: se vera si rimane nel ciclo, se false si interrompe il for;

  – incremento (o comunque modifica) della variabile.

Altro esempio di costrutto For è il **For...in**

```
for(<indice> in <oggetto>)
<istruzione>|{<blocco di istruzioni>}
```

Esso è utilizzato per analizzare le proprietà dell'oggetto indicato.

- **While**:

```
while (<condizione>)
<istruzione>|{<blocco di istruzioni>}
```

Il controllo di flusso While esegue le istruzioni finché la condizione è verificata.

Se la condizione non è mai verificata le istruzioni all'interno del ciclo non vengono eseguite neanche una volta.

- **do...While**:

```
do
 <istruzione>|{<blocco di istruzioni>}
while (<condizione>)
```

A differenza del costrutto While, il do...While esegue il blocco di istruzioni almeno una volta, dopo di che continua ad eseguire le istruzioni del ciclo finché la condizione è vera

2.6.3 Funzioni

Un programma solitamente ha la necessità di ripetere porzioni di codice in luoghi diversi. Ripetere le stesse istruzioni in luoghi diversi porta a errori. Per risolvere questo inconveniente si utilizzano le **funzioni**.

Le funzioni sono dei "contenitori" che racchiudono del codice, il quale può essere eseguito ogni qual volta che la funzione viene chiamata.

```
function square(x){
    return x * x;
}
square(5);
--> 25
```

La parola chiave "function" è sempre utilizzata quando viene effettuata la creazione della funzione. Essa è seguita dal nome della funzioni, che rappresenta il nome con cui la funzione viene salvata in memoria.

La parola chiave "return", seguita da un'espressione, consente di determinare il valore che la funzione restituisce. Quando il flusso di esecuzione si imbatte nella dichiarazione "return", esce dalla funzione corrente e restituisce il valore determinato dall'espressione.

Una delle particolarità che caratterizzano le funzioni in Javascript consiste nel poter "chiamare" una funzione (per esempio printer()) nonostante la funzione sia stata dichiara successivamente.

```
print("Print result: ", printer());

function printer(){
    return "RESULT printed";
}
```

All'avvio del programma, prima di ogni altra esecuzione, vengono individuate tutte le funzioni definite all'interno del programma e vengono memorizzate le relative associazioni, dopo di che si continua con l'esecuzione del resto del programma.

In questo modo non occorre preoccuparsi dove vengono definite e utilizzate le funzioni all'interno di un programma.

Quando si parla di funzioni, occorre introdurre due concetti fondamentali:

- **"scope"** (o anche **"lexical scope"**)

  Lo "scope" rappresenta il contesto di esecuzione all'interno del quale una particolare funzione JavaScript viene eseguita.

  Quando viene definita una funzione, essa viene costruita all'interno di un contesto di esecuzione superiore, ma allo stesso tempo, si viene a creare un contesto di esecuzione suo interno.

  Se all'interno della funzione vengono dichiarate delle variabili, esse hanno validità solo all'interno del contesto della funzione, sono cioè Variabili Locali che hanno validità solo all'interno della funzione. Diversamente, è possibile "vedere" da dentro la funzione tutte quelle

variabili che fanno parte dello stesso contesto di esecuzioni su cui è stata dichiarata la funzione.

```
var varTop = "top level";
var varLocal  = "local"
function parentFunction(){
    var varLocal = "local Function";
    function childFunction(){
        print(varLocal);
        print(varTop);
    }
    childFunction();
}
parentFunction();
print(varLocal);
--> "local Function"
      "top level"
      "local"
```

- **"closure"**

  Una delle più potenti feature di JavaScript è la possibilità di usare un particolare tecnica di mantenimento dello stato chiamata "closure".

  Il concetto di "closure" non è stato introdotto da JavaScript, altri linguaggi di programmazione ne fanno uso, tuttavia in JavaScript essa è realizzata mediante l'utilizzo del concetto di "inner function".

  Una "inner function" è una funzione che viene definti all'interno di un'altra funzione

  ```
  function outer_function(){
      function inner_function(){

      }
  }
  ```

  Le "inner function", essendo create nel local scope di una funzione, viene eliminata (garbage collected) non appena termina l'esecuzione

della funzione padre. Questo vuol semplicemente significare che non è possibile, dall'esterno dalla funzione padre, usare la funzione interna.

Una "inner function", però, facendo parte del contesto di esecuzione della funzione padre, può accedere direttamente al local scope della funzione padre, ovvero a tutte le sue variabili locali.

```
function outer_function(){
    var response = "result";
    function inner_function(){
     print(response);
    }
}
```

È proprio questo comportamento che definisce il termine "closure", lo scope della "inner function" si chiude/abbraccia lo scope della funzione padre.

Ma cosa accade se valorizzo una variabile globale con una "inner function" anonima?

```
var global_var;
function outer_function(){
    global_var = function (){

    }
}
```

In questo caso, collegando il global scope con il local scope di una funzione si fa in modo che quest'ultimo non venga deallocato, ma sopravviva all'eliminazione della funzione padre. Da questo ne consegue che la "inner function" sarà ancora utilizzabile.

Tecnicamente tutte le volte che JavaScript esegue una funzione vengono allocati internamente due oggetti: uno è un riferimento alla funzione stessa, e l'altro è un oggetto di tipo scope, il quale viene rilasciato alla fine della funzione, a meno che non sia stato collegato/referenziato da un'altro oggetto. Dunque non solo sarè possibile usare la "inner function" dall'esterno della funzione padre, ma sarà anche possibile accedere alle variabili del local scope interna ad essa.

```
var show_counter;
function outer_function(){
    var counter = 0;
    show_counter = function (){
     print(counter);
    }
}
```

Avere accesso ad un scope, ovviamente può anche voler dire essere in grado di modificarlo.

```
function outer_function(){
    var counter = 0;
    show_counter = function (){
     print(counter);
    }
    increase_counter = function(){
        counter++;
    }
}

outer_function();
show_counter();    // --> "0"
increase_counter();
show_counter();    // --> "1"
```

Il fatto che lo scope dove la "inner function" è stata dichiarata non viene deallocato, implica che le variabili, nel momento in una viene usata la "inner function", hanno il valore raggiunto al termina della funzione padre. Nell'esempio appena fatto all'interno della funzione vengono valorizzate delle variabili senza prima essere state dichiarate "var", questo sottintende la creazione delle variabili all'interno del global scope (ovvero come proprietè dell'oggetto window).

2.6.4 Oggetti

Un linguaggio di programmazione Object-Based come JavaScript solitamente, oltre ai valori di tipo primitivo, consente l'utilizzo anche di tipi composti, come gli **oggetti**.

I "numeri", le "stringhe" e i "boolean" sono valori di tipo primitivo che sembrano oggetti, perché essi hanno metodi, ma sono immutabili, al contrario degli oggetti che sono una "collezione" mutevole di dati.

In JavaScript, gli array sono oggetti, le funzioni sono oggetti, espressioni sono oggetti, e naturalmente, gli oggetti sono oggetti.

Un oggetto è un contenitore di dati, ognuno dei quali ha un nome e un valore. Il nome di un dato può essere una qualsiasi stringa, incluso la stringa vuota. Il valore di un dato contenuto in un oggetto, invece, può essere un qualunque valore JavaScript ad eccezione del valore "undefined".

Oltre ai valori, gli oggetti possono essere caratterizzati da metodi per la manipolazione degli stessi dati. In sostanza, i dati e le procedure che operano sugli stessi sono accorpati in un'unica entità detta oggetto.

Gli oggetti in JavaScript sono "class-free", cioè le classi non esistono e non ci sono vincoli di struttura o di comportamento stabiliti dalle classi.

Per chi è abituato alla programmazione ad oggetti in altri linguaggi, come ad esempio Java, è normale considerare gli oggetti come istanze di una classe, in JavaScript, come si è specificato in precedenza, non è così.

```
function Person(nome, cognome, eta){
    this.nome = nome;
    this.cognome = cognome;
    if(eta>0){
        this.eta = eta;
    }
}

var persona = new Persona("Mario","Rossi",23);
```

L'esempio proposto qui sopra mostra come creare un semplice oggetto JavaScript.

La prima cosa interessante da notare è che per creare un oggetto si è partiti dalla definizione di funzione.

Il compito che nei linguaggi Class-Based è svolto dalle classi, cioè definire gli attributi e i metodi di un oggetto, nei linguaggi Object-Based è svolto dalle funzioni.

La funzione utilizzata per la creazione di un oggetto è detta **costruttore**.

L'altra particolarità riguarda la parola chiave "this", essa all'interno di un costruttore si riferisce sempre al particolare oggetto che si sta creando.

Prendendo in considerazione l'esempio appena visto, si viene a creare un oggetto di nome *p*ersona, che possiede tre attributi: *nome*, *cognome* e *eta*, tutti e tre impostati da parametri passati alla funzione costruttore.

Ogni oggetto possiede una propria copia "personale" dei valori degli attributi che lo caratterizzano.

Il costruttore, come si è visto, permette di assegnare valori predefiniti o passati alla funzione agli attributi che possiede. Ma una volta creato l'oggetto i valori dei suoi attributi possono essere modificabili individualmente e possono assumere valori diversi da qualsiasi altro oggetto creato dalla stessa funzione costruttore.

Questo concetto si esprime affermando che gli oggetti, anche se creati mediante la stessa funzione costruttrice, hanno una proprietà "identità".

```
var persona2 = new Persona("Mario","Rossi",23);
```

In questo modo creiamo un secondo oggetto e con la sintassi

```
persona2.eta = 30;
```

accediamo all'attributo *eta* dell'oggetto con nome *persona2* e ne modifichiamo il valore.

Oltre ad attributi un oggetto può possedere **metodi**.

Un metodo è una funzione che opera sugli attributi di un oggetto determinandone il comportamento o le funzionalità.

```
function passanoGliAnni(eta){
    this.eta = eta;
}

function Person(nome, cognome, eta){
```

```
    this.nome = nome;
    this.cognome = cognome;
    if(eta>0){
        this.eta = eta;
    }
}
```

Il metodo *passanoGliAnni* ha la funzione di modificare l'età associata all'oggetto persona.

L'utilizzo della parola chiave "this" all'interno di un metodo è analogo a quello visto nella funzione costruttore: si riferisce cioè al particolare oggetto su cui il metodo è invocato.

La sintassi per invocare un metodo è la seguente

```
nome_oggetto.nome_metodo(argomenti_metodo);
```

nell'esempio fatto precedentemente

```
var persona = new Persona("Mario","Rossi",23);
persona.passanoGliAnni(24);
```

Un metodo può anche non avere argomenti.

```
nome_oggetto.nome_metodo();
```

La procedura appena descritta per creare metodi, anche se funzionante, non è delle migliori. Infatti abbiamo posto creato la funzione del metodo esternamente, allo stesso livello del costruttore dell'oggetto. Questo ci preclude la possibilità di creare un'ulteriore metodo, con lo stesso nome, per un oggetto diverso, perché si verrebbe a creare un conflitto sui nomi delle funzioni.

La soluzione consiste nell'incorporare la definizione del metodo all'interno della funzione costruttore.

```
function Person(nome, cognome, eta){
    this.nome = nome;
    this.cognome = cognome;
    if(eta>0){
        this.eta = eta;
    }
```

```
    this.passanoGliAnni = function(eta){
        this.eta = eta;
    }
}
```

Così facendo è possibili utilizzare lo stesso nome in altri metodi di altri oggetti o funzioni.

JavaScript oltre a consentire la creazione e manipolazione di nuovi tipi di oggetti, fornisce un insieme di **oggetti di sistema** facenti già parte del linguaggio:

- String: manipolare porzioni di testo memorizzato;
- Date: lavorare con date e orario;
- Array: memorizzare valori multipli all'interno di una sigola entità;
- Boolean: convertire un valore "non-Boolean" in un valore Boolean (true o false);
- Math: eseguire operazioni matematiche.

Ultimo argomento importante da trattare sugli oggetti JavaScript è quello dell'**ereditarietà**.

Come è stato detto JavaScript non essendo un linguaggio di programmazione Object-Based e non Class-Based non possiede il concetto di classe, di conseguenza come è possibile definire classi base e classi derivate?

Ogni funzione JavaScript possiede un **attributo prototype** che si riferisce ad un **oggetto prototype**.

All'oggetto prototype è possibile aggiungere attributi e metodi, ma occorre ricordare che non è come tutti gli altri oggetti, in quanto la sua funzione è fare da "modello" (prototipo appunto) per la creazione di altri oggetti.

Ogni attributo e metodo aggiunto ad un oggetto prototype viene reso disponibile a tutti gli oggetti della funzione costruttore a cui l'oggetto prototype è attribuito. Anche gli oggetti di sistema (come gli array) sono creati da funzioni costruttore che espongono un attributo prototype.

```
var a = new Array(3, 2, 6);
Array.prototype.sum = function(){
    it(typeof(this[i]) == "numbr"){
       r += this[i];
    }
    return r;
}
document.write(a.sum());    // --> 11
```

In questo esempio è stato aggiunto all'oggetto Array un metodo sum() che ritorni la somma di tutti gli elementi dell'array, ignorando eventualmente elementi non numerici.

Come si vede, anche oggetti già creati acquisiscono il nuovo metodo. Nell'esempio appena visto è stato aggiunto un singolo metodo ad un oggetto prototype, ma niente vieta di rimpiazzare completamente un oggetto prototype con un nuovo oggetto appositamente creato, cioÃ¨, data una funzione *F*

```
function F(){
    ....
}
```

ed un oggetto *p*, se

```
F.prototype = p;
```

gli oggetti che saranno, o sono stati già, creati utilizzano la funzione F come costruttore, avranno disponibili, e potranno utilizzare come propri, tutti gli attributi e metodi di *p*.

Implementazione del costruttore dell'oggetto *Person*.

```
function Person(nome, cognome, eta){
    this.nome = nome;
    this.cognome = cognome;
    this.passanoGliAnni = function(eta){
        if(eta>0){
            this.eta = eta;
        }
```

```
    }
    this.passanoGliAnni(eta);
}
```

Implementazione delle funzione costruttore con gli attributi specifici per la *persona Student*.

```
function Student(scuola,eta){
    this.scuola = scuola;
    this.passanoGliAnni(eta);
}
```

Sostituiamo l'oggetto prototype di *Studente* con un nuovo oggetto creato con la funzione *Person*

```
Student.prototype = new Persona();
var alunno = new Aereo("Liceo",16);
```

In conclusione, ogni oggetto creato da *Student* eredita tramite "prototype" attributi e metodi definiti nella funzione costruttrice *Person*.

Mediante l'utilizzo degli "Oggetti Prototype" si è riusciti a non perdere la caratteristica importante dell'ereditarietà, comune ai linguaggi di programmazione Class-Based.

2.7 Eventi

JavaScript è un linguaggio di scripting che viene utilizzato per rendere dinamiche e interattive le pagine Web e per superare i limiti congeniti dell'Html.

Il linguaggio Html è stato creato per la pubblicazione di documenti sul World Wide Web e non prevede comandi o tag che consentano l'interazione con l'utente (se non nella forma basilare del link e dei moduli) o con altri strumenti di web publishing.

Con l'Html è possibile pubblicare documenti che abbiano un titolo, una formattazione particolare, delle foto. Tuttavia non è possibile, per esempio, far eseguire al browser compiti specifici, come controllare le azioni dell'utente, utilizzare costrutti propri della programmazione tradizionale, cioè compiti specifici di JavaScript.

Mediante l'utilizzo di JavaScript, per avere una reale interattività con il mondo esterno occorre utilizzare il meccanismo degli **eventi**.

JavaScript è un *linguaggio di programmazione event-driven*, cioè un tipo di programmazione che supporta il meccanismo degli eventi.

La programmazione ad eventi è un paradigma fondamentale di programmazione dell'informatica. Mentre all'interno di un "programma tradizionale" l'esecuzione delle istruzioni segue percorsi fissi, predefiniti dal programmatore, nei programmi scritti utilizzando la tecnica a eventi, il flusso del programma è largamente determinato dal verificarsi di eventi, dovuto dall'interazione con l'esterno.

Nel seguente contesto, cioè quello di JavaScript, un evento è un "qualcosa" che si verifica durante l'interazione fra l'utente e la pagina: per esempio un click su un link, scorrimento della finestra del browser, insomma, qualsiasi tipo di interazione, non statica, con la pagina Web.

Utilizzano questo tipo di programmazione, i browser generano un evento ogni volta che succede qualcosa di interessante al documento o a un suo elemento. Ad esempio, il browser Web genera un evento quando finisce il caricamento di un documento, quando l'utente sposta il mouse sopra un collegamento ipertestuale, o fa clic sul pulsante di invio di un modulo.

Grazie all'ampia diffuzione e adozione dello standard DOM (Document Object Model), l'accesso agli elementi Html da parte del codice JavaScript funziona in modo molto simile in tutti i browser.

La combinazione dell'uso del DOM (Document Object Model) e degli eventi JavaScript sono un'unione fondamentali per la realizzazione di tutte le moderne applicazioni Web.

Da quanto i browser hanno iniziato a supportare JavaScript, hanno sviluppato un modello piuttosto semplice di gestione degli eventi, chiamato **event-handler**, e tutti i browser odierni offrono per questi handler un supporto compatibile, nonostante il fatto che per questi elementi non sia mai stato scritto uno standard definitivo.

Prima di vedere nello specifico il funzionamento e la gestione degli eventi, occorre soffermarsi su alcune definizioni di base.

- *event type* rappresenta la stringa che specifica il tipo di evento che si verifica. Il tipo "mousemove", per esempio, indica che l'utente ha spostato il mouse; il tipo "keydown" descrive a pressione di un tasto della tastiera; e il tipo "load" informa che un documento (o qualche

altra risorsa) ha terminato il caricamento dalla rete. Poiché il tipo di un evento è solo una stringa, spesso viene chiamato *event name*, anzi, tale nome viene utilizzato per specificare il genere di evento che si sta trattando.

- *event target* rappresenta l'oggetto/elemento su cui si è verificato l'evento o anche a cui è associato l'evento. Quando si parla di un evento, è necessario specificare a qual'è l'event target associato a quell'evento. Per esempio, l'evento "load" su un oggetto "window", oppure l'evento "onClick" su un'oggetto "button".

  Gli oggetti "window", "document" ed "element" sono gli obiettivi degli eventi più comuni in applicazioni JavaScript, ma alcuni eventi sono attivati su altri tipi di oggetti.

- *event handler*(gestione degli eventi) o *event listener* è una funzione che gestisce o risponde ad un evento. Ogni applicazione registra le proprie funzioni di gestione degli eventi mediante il browser, specificando il tipo dell'evento e l'oggetto/elemento a sui è associato l'evento.

  Quando un evento di un tipo specifico si verifica su un determinato oggetto/elemento, il browser richiama il gestore degli eventi. Quando un event handler viene invocato, a volte ci informa se il browser ha "sparato", "innescato" o "inviato" l'evento.

- *event object* è un oggetto associato ad un evento il quale contiene informazioni sul tale evento.

  Gli event object vengono passati come argomenti alle funzioni per la gestione degli eventi. Tutti gli event object hanno una proprietà che specifica il tipo di evento e una proprietà che specifica il destinatario dell'evento.

  Ogni tipologia di evento possiede un insieme di proprietà che la caratterizzano. Per esempio, l'event object associato all'evento del mouse comprende le coordinate del puntatore del mouse, l'event ojbect associato ad un evento tastiera contiene i dettagli sul tasto che è stato premuto.

  Molti eventi, però, definiscono solo le priprietà standard, come la tipologia e l'oggetto a cui è associato, questo perché non ha la necessità

di fornire altre informazioni utili. Per questi eventi l'importante è il semplice verificarsi dell'evento e non i dettagli dell'evento.

- *event propagation* è il processo mediante il quale il browser decide quale oggetto ha attivato l'event handler.

  Per gli eventi che sono specifici di un singolo oggetto (come ad esempio l'evento "load" di "window"), non è richiesta la propagazione. Invece, quando certi tipi di eventi si verificano su elementi del documento, si propagano a "bubble" (cioè a bolla) sulla struttura ad albero del documento.

  Per gestire questo procedimento, a volte è più conveniente registrare un singolo gestore degli eventi su un "documento" o un elemento contenitore piuttosto che su ogni singolo elemento interessato.

  Oppure, un event handler può bloccare la propagazione di un evento, in modo tale da non continuare l'effetto "bolla" e non innescare i gestore degli eventi delle strutture superiori. Questo è possibile farlo invocando un metodo specifico oppure impostando tale proprietà sull'event object.

  Un'altre forme di event propagation sono note come *event capturing*, i quali sono gestori appositamente registrati su elementi contenitori i quali hanno la possibilità di intercettare (o "catturare) gli eventi prima di essere consegnati al loro reali oggetto/elemento. Un esempio, è data dalla capacità di "catturare" gli eventi del mouse necessari durante l'elaborazione degli eventi di "trascinamento" del mouse.

Gli eventi in JavaScript sono piuttosto singolari. JavaScript, per gestire gli eventi, fa uso del pattern Observer o anche detto Event-Listener. L'aspetto fondamentale del funzionamento di questo pattern sta nell'uso delle funzioni di callback ogni qualvolta che lo stato di un determinato oggetto, nel nostro caso il verificarsi di un evento su un determinato elemento, cambia. Gli eventi in JavaScript operano in modo completamente asincrono, mediante l'uso di un unico thread che funge da Observer nei confronti degli eventi che possono verificarsi. Ciò significa che tutto il codice delle applicazioni sarà affidato ad azioni - come il un click dell'utente o il caricamento di una pagine - attivate dal thread principale che funge da Observer.

La fondamentale differenza che c'è fra la progettazione di "threaded program" e la "asynchronous program" si basa sul modo in cui si aspetta che qualcosa accada.

Nella "threaded program" occorre continuare a controllare ripetutamente se la condizione per l'avvento di un determinato evento è soddisfatta o meno. Mentre, nella "asynchronous program" occorre "registrare" una semplice funzione di callback con il relativo event-handler (gestore degli eventi), utilizzando questo modalità, ogni volta che l'evento si manifesta, il gestore degli eventi informa l'avvenuto evento mandando in esecuzione la funzione di callback.

Gli eventi JavaScript sono sostanzialmente legati agli elementi della pagine Web e al browser. In poche parole, ogni volta che un eventi di interesse, di un determinato elemento DOM, si verifica, si desidera una funzione che gestisca l'accaduto. Questo significa che è possibile fornire un riferimento a una porzione di codice che si vuole eseguire quando è necessario, per tutto il resto è il browser che si prende cura di tutti i dettagli.

2.7.1 Tipi di Eventi

Nei primi tempi del Web, i programmatori client-side facevano uso solo di un piccolo insiemi di eventi: "load", "click", "mouseover" e altri eventi simili. Queste tipologie di eventi sono ancora oggi ben supportate da tutti i browser.

Con la crescita della piattaforma Web sono simultanealmente cresciute le API che includono un insieme sempre più potente e vasto di eventi.

Al momento, non esiste nessun tipo di standard che definisce l'insieme completo degli eventi, e ad oggi il numero degli eventi supportati dai vari browser sta crescendo rapidamente.

Lo sviluppo di nuovi eventi proviene da tre principali fonti distinte:

- dalle specifiche del DOM Level 3 Events, che dopo un lungo periodo di inattività a ripreso il lavoro sotto la supervisione del W3C.

- molte nuove API nella specifica dell'Html5 definiscono nuovi eventi per attività come la gestione dell'audio e del video etc.

- l'avvento dei dispositivi touch-based e JavaScript-abilitati come per esempio l'IPhone, hanno richiesto la definizione di nuovi eventi per la gestione e per il "touch".

Occorre tenere presente, che molti di questi nuovi tipi di eventi non sono ancora stati ampiamente implementato o sono definiti da specifiche ancora in fase di progettazione.

Gli eventi più comunemente usati all'interno delle applicazioni web, su cui ruotano la maggior parte delle interazioni, e che sono universalmente supportati da tutti i browser sono:

- **Form Events**
- **Windows Events**
- **Mouse Events**
- **Key Events**

Oltre agli eventi classici, descritti qui sopra, attualmente vi sono diverse fonti di sviluppo di nuovi eventi, fra queste vi sono le specifiche del DOM Level 3 Events.

Le specifiche del DOM Level 3 Events sono arrivate ad oggi dopo circa una decina di anni di sviluppo da parte del W3C.

Ultimamente queste specifiche sono state revisionate per renderle compatibili con le esigenze correnti del Web. Ad oggi si è arrivati ad un'ultima fase di progetto, cioè alla standardizzazione di queste nuove specifiche degli eventi. Quest'ultima fase ha standardizzato la maggior parte degli eventi "storici" sopra descritti, e ne ha aggiunto di nuovi.

I nuovi eventi, introdotti nell'ultima specifica, non sono ancora ampiamente supportati, ma una volta concluso lo standard, i produttori di browser saranno tenuti ad effettuare le modifiche per supportare tutti gli eventi della specifica.

Oltre alla specifiche del DOM Level 3 Events, anche Html5 ha introdotto nuove specifiche per la definizione di nuovi eventi.

Html5 ha definito una serie di nuove API per lo sviluppo delle applicazioni Web, fra queste ve ne sono molte per la definizione di nuovi eventi:

- Una delle caratteristiche ampiamente pubblicizzate dell'Html5 si base sull'introduzione di due nuovi elementi: "audio" e "video", per la riproduzione di suono e video.

  Questi elementi hanno una lunga lista di eventi che fanno scattare l'invio di notifiche per quanto riguarda gli eventi di rete (network events), i dati sullo stato del buffering, lo stato di riproduzione ect.

- Le API Html5 per il "drag and drop" consentono alle applicazioni JavaScript di trasferire dati tra un'applicazione Web e applicazione nativa.

  Gli eventi drag and drop vengono attivati in maniera simile agli eventi del mouse. Associato all'evento vi è la proprietà "dataTranfer" che contenente informazioni sui dati che vengono trasferiti e sul formato disponibile.

- Html5 include il supporto per gestire le applicazioni web offline.

  I due eventi più importanti associati a questo supporto sono gli eventi "online" e "offline": essi si attivano ogni volta che la finestra del browser perde o aggancia la connessione in rete.

- Html5 prevede delle API per il Web Storage, le quali definiscono l'evento storage, il quale prevede notifiche riguardanti le modifiche sui dati memorizzati.

Con la diffusione di dispositivi mobile, in particolari quelli touchscreen, è nata l'esigenza di creare nuove categorie di eventi.

In molti casi, gli eventi touchscreen vengono mappati in tradizionali tipi di evento come un click e lo scroll. Ma non ogni interazione con un'interfaccia utente touchscreen può essere paragonata ad un mouse, e non tutti i "tocchi" possono essere trattati come gli eventi del mouse. Al momento, non ci sono standard per gli eventi touchscreen, ma il W3C ha iniziato a lavorare su "Touch Events Specifications" che utilizzano gli eventi touch di Apple come punto di partenza.

Il continuo sviluppo di nuovi eventi e specializzazioni di eventi già esistenti permettono alle applicazioni Web di rimanere al passo con le esigenze degli utenti.

La gestione di eventi nuovi e sempre più aggiornati, porta ad avere applicazioni Web sempre più innovative e basate sull'interazione con l'utente, caratteristica fondamentali delle moderne applicationi Web.

2.7.2 Registrazione dell'Event Handler

Come associare i gestori degli eventi agli elementi Html è stata una ricerca in continua evoluzione in JavaScript.

Tutto è cominciato con i primi browser, che obbligava l'implementatore a scrivere il gestore degli eventi in linea con il codice Html.

Con il passare del tempo, tale obbligo è diventato sempre meno popolare e la tecnica è stata raffinata.

Nel periodo in cui Netscape e Internet Ecplorer entrarono in competizione fra loro, ognuno di essi sviluppò due distinti, ma molto simili, modelli di registrazione degli eventi. Ad uscirne "vincitore" fu Nascape, infatti in suo modello divenne uno standard del W3C.

Ad oggi, sono presenti tre tipi di modelli affidabili per la registrazione degli eventi:

- **Metodo tradizionale**
- **Metodo W3C**
- **Metodo IE (Internet Explorer)**

Il **metodo tradizionale** di registrare gli eventi è di gran lunga il più semplice e il più compatibile modo di gestire gli eventi. Per utilizzare questo approccio occorre allegare una funzioni alla "proprietà evento" dell'elemento DOM che si desidera controllare.

```
// ricerca il primo elemento <form> e allega ad esso
// il 'submit' event handler
document.getElementsByTagName("form")[0].onSubmit = function(e){
    return stopDafault(e);
};
// allega 'keyPress' event handler all'elemento <body> del documento
document.body.onKeyPress = myKeyPressHandler;
// allega 'load' event handler alla pagina
window.onLoad = function(){ ... };
```

Questa particolare tecnica ha una serie di vantaggi e di svantaggi, di cui si deve essere consapevoli nel momento in cui si utilizza.

Vantaggi:

- il più grande vantaggio riguarda l'incredibile semplicità nell'utilizzo e la coerenza, nel senso che si è praticamente sicuri che funzioni indipendentemente dal browser che si utilizza;

- nel gestire un evento, se si utilizza la parola chiave "this" essa si riferisce all'elemento corrente che si sta trattando.

```
var li = document.getElementsByTagName("li");
for(var i=0; i<li.length; i++){
    li[i].onClick = handleClick;
}
function handleClick(){
    this.style.backgroundColor = "blue";
    this.style.color = "white";
}
```

Svantaggi:

- il metodo tradizionale lavoro solo con eventi che utilizzano la propagazione "bubble", non con eventi che utilizzano la propagazione "capture" e "bubble";

- è possibile associare un solo event handler alla volta per ogni elemento. Ciò causa una potenziale confusione quando si lavora con la popolare proprietà window.onLoad.

```
// registro il primo handler
window.onLoad = myFirstHandler;
// da qualche parte, in un'altra libreria inclusa
// il primo handler viene sovrascritto
// solo 'mySecondHandler' viene chiamato
// quando la pagina finisce il caricamento
window.onLoad = mySecondHandler;
```

Il **metodo del W3C** per la registrazione degli event handler sugli elementi del DOM è l'unico mezzo veramente standardizzato per eseguire questa operazione. Questo implica che tutti i browser moderni supportano questo metodo di associare gli eventi, ad eccezione, ovviamente, di Internet Explorer.

Il codice per associare una nuova funzione handler è molto semplice, ogni elemento DOM possiede una funzione, denominata addEventListener, che richiede tre parametri: il nome dell'evento, la funzione che gestirà l'evento, e un "flag" per attivare o disattivare la "capture" della propagazione dell'evento.

```
// ...
document.getElementsByTagName("form")[0].addEventListener('submit',
               function(e){ return stopDefault(e);},
               false);
// ...
document.body.addEventListener('keypress', myKeyPress, false);
// ...
window.addEventListener('load', function(){...}, false);
```

Vantaggi:

- questo metodo supporta sia la fase di "capture" che quella di "bubble" di gestione degli eventi. Questa fase degli eventi viene attivata impostando l'ultimo parametro di addEventListener su false ("bubble") o su true ("capture");
- all'interno della funzione di event handler, la keyword "this" fa riferimento all'elemento corrente che si sta trattando;
- l'event object è sempre disponibile nel primo parametro della funzione di handling;
- è possibile associare più event handler sullo stesso elemento, senza pericolo di sovrascrivere i gestori precedentemente associati.

Svantaggi:

- la funzione "addEventListener" non funzione su Internet Explorer, è necessario utilizzare la funzione "attachEvent".

Per molti aspetti, il **metodo Internet Explorer** utilizzato per registrare gli event handler sembra essere simile a quello adottato dal W3C. Tuttavia, quando si va a vedere i dettagli, i due metodi iniziano a differire su aspetti molto significativi.

```
// ...
document.getElementsByTagName("form")[0].attachEvent('onsubmit',
               function(e){ return stopDefault(e);});
// ...
document.body.attachEvent('onkeypress', myKeyPressHandler);
// ...
window.attachEvent('onload', function(){...});
```

Vantaggi:

- è possibile associare più event handler sullo stesso elemento, senza pericolo di sovrascrivere i gestori precedentemente associati.

Svantaggi:

- IE supporta solo la fase di "bubble" per la propagazione dell'evento;
- la parola chiave "this" all'interno di una funzione che gestisce l'evento si riferisce all'oggetto "window" e non all'elemento corrente che si sta trattando (questo è un'enorme svantaggio di IE);
- l'event object è disponibile sono nel parametro "window.event";
- il nome dell'evento deve essere un nome come "ontype";
- funziona solo su IE. Se si vuole utilizzare browser diversi da IE occorre utilizzare addEventListener del W3C.

Per sopperire alla differenza che c'è fra le tipologie di browser, è comune vedere

```
var b = document.getElementById("mybutton");
var handler = function() {  alert("Thanks!!");  };
// non-IE browser
if(b.addEventListener)
    b.addEventListener("click", handler, false);
```

```
// IE browser
else if(b.attachEvent)
    b.attachEvent("onclick", handler);
```

Le modalità appena analizzate associano l'event handler alla proprietà evento dell'elemento DOM in JavaScript.

Oltre alla possibilità dell'associazione tramite proprietà, vi è anche la possibilità di registrare l'event handler tramite gli attributi Html.

L'event handler viene registrato impostandolo come attributo del tag Html corrispondente.

```
<button onClick="alert('Hello World');">Click Here</button>
```

Se si esegue tale operazione di registrazione, il valore dell'attributo, in questo caso "onClick", deve essere una stringa corrispondete al codice JavaScript. Tale codice dovrebbe corrispondere al corpo della funzione dell'event handler, non la alla dichiarazione della funzione completa. Cioè non devono essere presenti le parentesi graffe precedute dalla dichiarazione "function".

Se l'attributo Html contiene più dichiarazioni JavaScript, è necessario separare queste dichiarazioni con il punto e virgola oppure interrompendo il valore dell'attributo separandolo su più linee.

Lo stile di programmazione la client, consiste nel mantenere il contenuto dell'Html separato dal comportamento di JavaScript. I programmatori che seguono questo stile di programmazione, evitano di registrare gli events handler sugli attributi Html, in quanto il contenuto JavaScript e Html andrebbe a mescolarsi.

Piuttosto, viene utilizzato il metodo di registrazione sulle proprietà degli elementi DOM, in tal modo i contenuti Html e JavaScript rimangono separati.

2.7.3 Invocazione dell'Event Handler

Una volta registrato l'event handler, il browser Web richiama (o invoca) automaticamente il gestore degli eventi quando un certo tipo di evento si verifica per un certo oggetto specifico.

Questa sottosezione descrive come il gestore degli eventi viene invocato, gli argomenti del gestore degli eventi, il contesto e l'ambiente dell'invocazione, e il significato del valore di ritorno di un gestore degli eventi.

Oltre a descrivere come i gestori degli eventi vengono richiamati, verrà mostrato anche come gli eventi si propagano.

I gestori degli eventi registrati sulle proprietà degli elementi del DOM, sono di solito invocati assieme ad un event object (salvo eccezioni), esso rappresenta l'unico e solo argomento associato all'event handler.

Le proprietà dell'event object forniscono i dettagli relativi all'evento. La proprietà "type", per esempio, specifica il tipo di evento che si è verificato.

In IE8 (Internet Explorer 8) e nelle versioni precedenti, gli events handler registrati su proprietà degli elementi DOM, quando invocati, non passavano nessun event object. L'event object era disponibile tramite la variabile globale "window.event".

Per la portabilità del codice fra i browser (cioè fra IE e tutti gli altri browser), è possibile scrivere l'event handler in questo modo, così da usare "window.event" nel caso in cui l'argomento event object non venga fornito:

```
function handler(event){
    event = event || window.event;
    // il codice dell'handler comincia qui
}
```

Gli event handler registrati con "attachEvent()" passano l'event object, ma possono utilizzare anche "window.event".

Quando si registra un event handler impostando un attributo Html, il browser converte il codice JavaScript dell'attributo in una funzione.

Browser diversi da IE costruiscono una funzione con un singolo argomento avente il nome dell'evento. IE, invece, costruisce una funzione che non prevede nessun argomento. Per identificare tale evento, si fa riferimento a "window.event".

In entrambi i casi, i gestore degli eventi Html possono utilizzare l'event object come riferimento all'evento.

Quando si registra un event handler sulla proprietà di un elemento DOM, è come se si stesse definendo un nuovo metodo per l'elemento del documento.

```
e.onclick = function() { /* codice handler */ };
```

Non sorprende, quindi, che i gestori degli eventi vengono invocati (con eccezione di IE) come metodi dell'oggetto su cui sono definiti. Cioè, all'in-

terno del corpo di un gestore di eventi, la parola chiave "this" si riferisce al destinatario dell'evento.

Ciò accade quando si effettua la registrazione mediante "addEventListener()". Sfortunatamente, questo, non è vero con l'utilizzo di "attachEvent()": i gestori registrati con "attachEvent()" vengono invocati come funzioni, e il valore di "this" fa riferimento all'oggetto globale "window".

Come tutte le funzioni in JavaScript, anche le funzioni degli event handler hanno un *lexical scope*. Esse vengono eseguite nell'ambito dove è presente il codice della funzione e non nell'ambito in cui vengono invocate, in questo modo, esse possono accedere a tutte le variabili locali nell'ambito di esecuzione.

Rappresentano un caso particolare i gestori degli eventi registrati sugli attributi Html.

Tali event handlers vengono convertiti dal browser in funzioni di "top-level" così da avere accesso a tutte le variabili globali, ma non a tutte le variabili locali.

```
function(event){
    with(document){
        with(this.form) || {}){
            with(this){
              /* codice */
            }
        }
    }
}
```

I gestori degli eventi registrati su attributi Html, possono utilizzare l'object target, l'oggetto "form" (se esiste) e l'oggetto "document", così come se essi fossero variabili locali.

Gli attributi Html non sono luoghi naturali per includere lunghe stringhe di codice, per questo, grazie alla "scope chain" c'è la possibilità di usare "scorciatoie".

Per esempio, è possibile utilizzare "TagName" piuttosto che "this.TagName", oppure "getElementById" al posto di "document.getElementById", infine per gli elementi che si trovano all'interno della "form", è possibile riferir-

si a qualsiasi elemento della "form" utilizzando l'"ID" dell'elemento (usare zipcod piuttosto che this.form.zipcode).

D'altra parte, la "scope chain" di gestione di eventi Html è sorgente di inconvenienti, perché le proprietà di ogni oggetto vengono "oscurate" da proprietà di oggetti globali aventi lo stesso nome.

Il valore di ritorno di un event handler, registrato sulla proprietà di un'elemto DOM oppure su un attributo Html, è un qualcosa di molto significativo.

In generale, un valore di ritorno "false" dice al browser che non deve essere eseguita l'azione predefinita associata all'evento.

Per esempio, il gestore "onclick" di un pulsante "submit" all'interno di una "form", può restituire "false" per evitare che il browser invii il modulo, questo può accadere quando l'input dell'utente non è corretto.

Sostanzialmente il valore di ritorno di un gestore degli eventi viene utilizzato in caso di validazione, per effettuare controlli lato client ogni qualvolta che si effettua un'operazione di interazione.

Prendendo per assodato che un elemento del documento o un'altro oggetto può avere più di un event handler registrato per un determinato tipo di evento, come viene gestito l'ordine di invocazione degli event handler quando quel determinato evento si verifica?

Quanto tale evento si verifica, il browser deve invocare tutti i gestori degli eventi, seguendo le seguenti regole di ordine di invocazione:

- gli event handler registrati su proprietà o attributi Html, eventualmente, sono sempre i primi a essere invocati;
- gli event handler registrati con "addEventListener()" vengono invocati nell'ordina con cui sono stati registrati;
- gli event handler registrati con "attachEvent()" possono essere invocati in qualsiasi ordine e il codice non dovrebbe dipendere dall'invocazione sequenziale.

Queste regole fanno capire che occorre fare attenzione in che punto o meno vengono effettuate le registrazioni degli event handler.

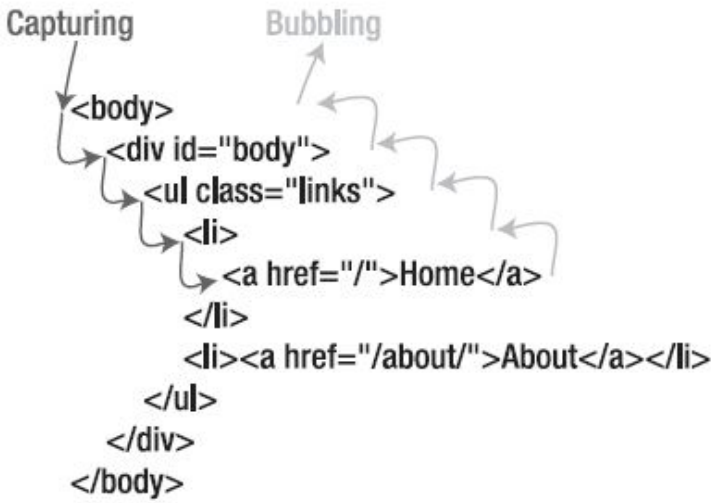

Figura 2.2: Fasi di Capturing e Bubbling

Gli eventi, quando vi verificano, sono caratterizzati da un "event flow" che si propaga da un punto di partenza ad uno d'arrivo.
Quando il destinatario di un evento è l'oggetto "window", o qualche altro oggetto autonomo, il browser risponde ad un evento semplicemente invocando gli appropriati gestori degli eventi su tale oggetto.

Quando, però, il destinatario dell'evento è "document" o un elemento di esso, tuttavia, la situazione è più complicata. Questo perché il verificarsi di un determinato evento genere un flusso attraverso la pagina stessa.

Questo flusso è il processo attraverso il quali un qualsiasi evento generato in una pagina, percorre la struttura gerarchica del documento (DOM), fino ad arrivare all'object target dell'evento, e una volta raggiunto, ripercorre a ritroso il DOM fino ad uscirne.

In sostanza, il flusso dovuto al verificarsi dell'evento, è costituito da una fase di "andata" della **capturing** e una di "ritorno" detta **bubbling**.

È possibile vedere un esempio di ordine di esecuzione della Figura 2.2.

La figura mostra quale gestore degli eventi viene invocato, in quale ordine, ogni volta che un utente clicca sul primo elemento "a" della pagina.

Osservando l'esempio, è possibile verificare il flusso dell'evento. Se si finge che un ipotetico utente clicchi sull'elemento "a", per primo viene lan-

ciato l'handler del "document", dopo di che l'handler del "body", di seguito l'handler del "div", e così via, fino ad arrivare all'elemento "a", questa viene detta *fase di cattura* (capturing).

Una volta terminata l'invocazione, si ripercorre nuovamente a ritroso l'"albero" del DOM, attraverso gli handler di "li", "ul", "div", "body" e "document" lanciati precedentemente, in questo ordine.

Ci sono molti specifici motivi per cui la gestione degli eventi è costruita in tale modo e funziona così bene. Si osservi l'esempio seguente.

```
var li = document.getElementsByTagName("li");
for(var i=0; i<li.length; i++){
    li[i].onmouseover = function(){
        this.style.backgroundColor = 'blue';
    };
    li[i].onmouseout = function(){
        this.style.backgroundColor = 'white';
    };
}
```

Il seguente codice registra su ogni elemento "li" due event handler, uno per cambiare il colore dello sfondo quando il mouse si muove su tale elemento, e l'altro quando il mouse si sposta fuori dall'elemento.

Tale codice fa esattamente ciò per cui è stato dichiarato: quando il mouse va sopra l'elemento "li" il suo background cambia colore, ma quando il mouse esce da sopra di esso il colore ritorna com'era.

Tuttavia dal momento che l'elemento "li" contiene anche un elemento "a", quando ci si sposta da sopra l'elemento "li" si passa automaticamente sull'elemento "a" e viceversa.

Il flusso esatto di invocazione degli eventi è il seguente:

- "li" mouseover: il mouse si sposta sull'elemento "li";
- "li" mouseout: il mouse passa da "li' a "a" contenuto all'interno di essa;
- "a" mouseover: il mouse è ora sull'elemento "a";
- "li" mouseover: l'evento bolle mouseover di "a" sale fino al mouseover di "li".

È possibile notare dal modo con cui si stanno invocando gli eventi, che si sta ignorando completamente la fase di "capturing". Questo è dovuto al fatto che si è utilizzato il metodo tradizionale per associare l'event handler all'elemento, infatti, il metodo tradizionale supporta solo la fase di "bubbling" degli eventi, e non la fase di "capturing".

Nel seguente ordine, oltre ad aver ignorare la fase di capturing, vi sono altre due azioni particolari: il "mouseout" sull'elemento "li" e il "mouseover bubbling" dall'elemento "a" all'elemento "li".

L'evento "mouseout" si verifica perché, a giudizio del browser, si è lasciato l'elemento padre "li" per trasferirsi su di un'altro elemento, nonostante questo faccia parte del componente padre. Tutto ciò è dovuto al fatto che qualsiasi elemento riceve o meno il cosiddetto "focus" del mouse, che non dipende assolutamente dai rapporti di "parentela" che ci sono fra gli elementi.

Per quanto riguarda, invece, il fenomeno del "mouseover bubbling", dal momento che non vi è nessun tipo di vincolo che arresti l'effetto bubbling, l'evento che si verifica sull'elemento "a" continua semplicemente la sua propagazione sull'albero DOM, alla ricerca di altri elementi in ascolto.

Il primo elemento che si incontra nel processo di "bubbling" è l'elemento "li", che è in ascolto sull'evento "mouseover", cioè per l'entrata del mouse.

C'è modo di fermare la propagazione dell'effetto "bubbling"?

L'arresto della fase di "bubbling" è un'operazione che può rivelarsi estremamente utile in applicazioni complesse.

Sfortunatamente, IE offre una modalità diversa da tutti gli altri browser per fermare l'effetto "bubbling".

```
function stopBubble(e){
    // non-IE browser
    if(e & e.stopPropagation)
        e.stopPropagation();
    else // IE browser
        window.event.cancelBubble = true;
}
```

L'esempio seguente mostra come annullare l'effetto "bubbling". Tale funzione gestisce entrambi i metodi di cancellazione dell'effetto "bubbling": quello standard fornito da W3C, e quello non standard utilizzato da IE.

Con la possibilità di bloccare l'effetto "bubbling" sugli eventi, si ha il controllo completo sugli elementi e sulla gestione degli eventi. Si tratta di uno strumento fondamentalmente necessario per lo sviluppo di applicazioni web dinamiche.

L'effetto finale, infatti, è quello di annullare l'azione di default del browser, cioè ci permette di ignorare il comportamento classico del browser e ci permette di implementare nuove funzionalità, per creare vere e proprio applicazioni totalmente dinamiche e interattive.

Capitolo 3

Da JavaScript a Dart

3.1 Cos'è Dart

Il suo "motto" per eccellenza è "Structured Web Programming" e il suo nome è Dart.

Dart è il nuovissimo linguaggio pensato per la programmazione Web strutturata introdotto da Google, alla fine del 2011. Si tratta di un linguaggio strutturato, orientato agli oggetti, progettato per migliorare ed eventualmente sostituire l'utilizzatissimo JavaScript, o almeno per offrirne un'alternativa.

Nonostante Dart sia ancora in una fase iniziale, gli obbiettivi progettuali che muovono l'evoluzione di questo progetto open source sono:

- creare un linguaggio di programmazione web strutturato ma flessibile;
- rendere Dart un linguaggio di programmazione familiare e naturale ai programmatori e quindi facile da imparare;
- assicurarsi che Dart offra elevate prestazioni e avvii rapidi delle applicazioni;
- rendere Dart utilizzabile in qualunque ambiente e in qualsiasi tipologia di dispositivo Web, come telefoni, tablet e portatili;
- fornire strumenti per rendano Dart, utilizzabile su tutti i migliori browser moderni ora in "circolazione", ed eseguibile lato server;

Questi obiettivi di progettazione devono far fronte ai seguenti problemi attualmente presenti nella programmazione web:

- la mancanza di una struttura negli script, realizzati all'interno di applicazioni web di grandi dimensioni, portano a problemi di correzione e mantenimento del codice. Inoltre queste applicazioni non possono essere suddivise in modo tale da assegnare i vari compiti all'interno di una squadra di lavoro.

- i linguaggi di scripting sono popolari perché è nella loro natura la scrittura veloce di codice.

  In generale, l'approccio su porzioni di applicazioni, progettate da altri, sono possibile mediante l'utilizzo di commenti, piuttosto che nell'utilizzo intrinseco della struttura del linguaggio. Come risultato si evince la difficoltà, per un individuo diverso dall'autore, a leggere o mantenere una particolare porzione di codice.

- con i linguaggi esistenti, lo sviluppatore è costretto a fare una scelta tra linguaggi statici o dinamici. Tradizionalmente i linguaggi statici richiedono uno stile di codifica molto rigido e eccessivamente vincolato.

- gli sviluppatori non sono stati in grado di creare sistemi omogenei che comprendano sia la parte client che la parte server, fatta eccezione per alcuni casi particolare come Node.js e Google Web Toolkit (GWT).

- diversi linguaggi comportano "context switches", i quali sono ingombranti e aggiungono complessità al processo di codifica.

Dart viene sviluppato con gli obiettivi di essere semplice, efficiente e scalabile, un linguaggio di programmazione che combina nuove caratteristiche con costrutti di linguaggi familiari, in modo da rendere la sintassi chiara e leggibile.

3.2 Linguaggio Class-Based e Interfacce

Una delle particolarità più evidenti che differenzia Dart da JavaScript sta nell'uso delle **classi**.

JavaScript è un linguaggio di scripting Object-Based, cioè un linguaggio che si basa sugli *oggetti* e non prevede l'utilizzo delle *classi*.

Le classi sono dei costrutti utilizzati come modelli per la creazione degli oggetti. Ogni modello può contenere attributi e metodi.

Un oggetto creato a partire da una certa classe viene detto **istanza** di quella classe.

Associato al concetto di classe, vi è quello di **interfaccia**. Un'interfaccia rappresenta l'insieme dei dati e dei metodi visibili all'esterno degli oggetti che sono istanze di una classe.

Dart, a differenza di JavaScript, essendo un linguaggio Class-Based prevede i concetti di interfaccia, di classe e di oggetti che vengono istanziati a partire da esse.

```
interface Shape{
    num perimeter();
}
class Rectangle implements Shape{
    final num height, width;
    Rectangle(num this.height, num this.width);
    num perimeter() => 2*height + 2*width;
}
class Square extends Rectangle{
    Square(num size) : super(size, size);
}
```

Come viene mostrato nell'esempio, un'altra caratteristica fondamentale legata al concetto di classe è quella di **ereditarietà**.

L'ereditarietà permette di descrivere il legame che c'è fra le diverse classi. Essa consente di definire classi specializzate a partire da classi già definite.

Nel nostro esempio, la classe "Square" eredita dalla classe "Rectangle", questo perché uno rappresenta la specializzazione dell'altro, infatti un "quadrato" concettualmente rappresenta un caso particolare di "Rettangolo" con tutti i lati uguali.

Il concetto di ereditarietà permette la riusabilità e l'estensione di codice già presente.

Diversamente, JavaScript dato che non fa uso del concetto di classe, realizzare l'ereditarietà utilizzando un tipo di programmazione basata sui prototipi, meno intuitiva ed efficiente rispetto all'implementazione con l'uso delle classi.

```
class Welcome{
    var prefix = 'Hello, ';
    saluto(name){
        print('$prefix $name');
    }
}
main(){
    var welcome = new Welcome();
    welcome.saluto("World!!");
}
```

L'esempio seguente mostra le caratteristiche base di una classe Dart.

Le classi in Dart sono composte da costruttori, metodi e variabili i quali possono essere statici o istanze.

La dichiarazione *class* permette di definire una classe, chiamata "Welcome", che ha come default la superclasse *Object*.

In Dart, tutte le classe discendono direttamente o indirettamente da *Object*, questo è vero quando non viene applicata l'ereditarietà con la parola chiave *extends*. In quel caso si va a specificare la superclasse da cui si eredita, la quale non è più *Object* ma un'altra classe specificata.

Dart ha la particolarità di essere a **singola ereditarietà**, cioè le classe figlie possono ereditare da una sola classe padre o detta anche superclasse.

All'interno di una classe, le istanze delle variabili, proprio come normali variabili, possono essere create con l'uso di *var*, *final*, o specificando il tipo.

Nel nostro esempio, ogni oggetto "Welcome" ha una propria copia della variabile "prefix" inizializzata con "Hello, ". Le variabili istanziate all'interno di un oggetto, possono essere manipolate, direttamente utilizzando la notazione puntata

```
welcome.prefix = 'Hi, ';
```

attraverso i costruttori, oppure attraverso metodi *setter*.

I costruttori sono metodi che hanno lo scopo di creare le istanze delle classi e di inizializzarle durante il processo di creazione.

Per creare una nuova istanza di una classe occorre invocare una *new* seguita dal costruttore della classe che si vuole istanziare.

```
new Welcome();
```

Poiché nell'esempio fatto, all'interno della classe non è stato dichiarato nessun costruttore, di default viene richiamato il costruttore, senza argomenti, della superclasse.

Dato che "Welcome" non specifica nessun tipo di legame con un'altra classe, la sua superclasse è *Object*, e "new Welcome()" invocherà il costruttore di *Object*.

Oltre alle variabili, all'interno di una classe vi sono i metodi. Esso consentono di definire funzioni legate al particolare oggetto su cui sono richiamate, per questo vengono chiamati *istanze di metodi*.

Le classi in Dart possiedono molte altre caratteristiche aggiuntive. Quelle comunemente più utilizzate sono:

- *named constructors*
- *setter* e *getter*.

Quando una classe ha necessità di dichiare più di un costruttore, è possibile definire i *named constructors*. Essi permettono di assegnare ai vari costruttori un'"etichetta" in modo tale da differenziare la chiamata di uno o dell'altro costruttore.

```
class Welcome{
    var prefix = 'Hello, ';
    Welcome();
    Welcome.setPrefix(this.prefix);
    saluto(name){
        print('$prefix $name');
    }
}
main(){
    var welcome = new Welcome.setPrefix('Hi, ');
    welcome.saluto("World!!");
}
```

Nell'esempio seguente, oltre al costruttore di default, il quale può anche non essere dichiarato, vi è un'altro costruttore, aggiunto per soddisfare determinate condizioni imposte dall'implementatore.

Nel codice appena visto, l'abbreviazione "this.prefix" come argomento del costruttore, rappresenta una "scorciatoia" per assegnare un determinato valore alla variabili istanziata all'interno della classe.

Altra particolarità delle classi Dart sono *setter* e *getter*. *Setter* e *getter* rappresentano due modi per consentire la manipolazione delle variabili istanziate, senza accedere direttamente ad esse.

```
class Welcome{
    String _prefix = 'Hello, ';
    String get prefix() => _prefix;
    void set prefix(String value){
        if(value == null) value = "";
        if(value.length > 20) throw 'Prefix too long!';
        _prefix = value;
    }
    saluto(name){
        print('$prefix $name');
    }
}
main(){
    Welcome welcome = new Welcome();
    welcome.prefix = 'Hi, ';
    welcome.saluto("Setter!!");
}
```

In questo modo non si va a manipolare direttamente variabili proprie dell'ambiente dell'istanza dell'oggetto.

Cosa rappresenta il simbolo "_" prima della variabile "prefix" ?

Dart supporta due livelli di **privacy**: *pubblico* e *privato*. Per effettuare una dichiarazione privata occorre utilizzare il carattere "_", altrimenti la dichiarazione assume automaticamente valore pubblico.

La variabile "prefix" se dichiarata privata, come nell'esempio fatto, è accessibile sono dall'interno della classe, per essere accessibili direttamente dall'esterno deve essere dichiarata pubblica.

La privacy è un concetto statico legato a particolari porzioni di codice.

Essa è stata progettata per supportare problemi inerenti all'ingegneria del software piuttosto che per un problema di sicurezza.

A fianco del concetto di classe, per la natura del linguaggio Dart, vi sono le **interfacce**.

Un'interfaccia definisce come è possibile interagire con un oggetto. Essa contiene metodi, setter e getter, costruttori e un insieme di "superinterfacce".

All'interno delle interfacce vengono specificate soltanto le signature dei metodi, e non il "corpo", il quale viene poi realizzato all'interno della classe che implementa l'interfaccia.

Oltre ai metodi, un'interfaccia può contenere un insieme di "superinterfacce". Questo insieme è costituito da tutte quelle specifiche da cui eredita l'interfaccia che si va a creare.

In Dart le interfacce sono un elemento di fondamentale importanza, la gran parte delle librerie che stanno alla base di Dart sono definite in termini di interfacce.

```
class Welcome implements Comparable{
    String prefix = 'Hello, ';
    Welcome(){}
    Welcome.withPrefix(this.prefix);
    saluto(name) => print('$prefix $name');

    int compareTo(Welcome other) => prefix.compareTo(other.prefix);
}
main(){
    Welcome welcome = new Welcome();
    Welcome welcome2 = new Welcome.withPrefix('Hi, ');
    num result = welcome2.compareTo(welcome);
    if(result == 0){
        welcome2.saluto('you are the same.');
    }else{
        welcome2.saluto('you are different.');
    }
}
```

Il seguente esempio mostra l'implementazione dell'interfaccia "Comparable" della Core Library di Dart, da parte della classe "Welcome". Viene mostrata la semplice operazione di comparazione fra due oggetti "Welcome".

L'implementazione di una interfaccia è composta da due fasi. La prima consiste nell'aggiungere la dichiarazione "implements Comparable" alla dichiarazione della classe. La seconda consiste nel definire tutti i metodi previsti dall'interfaccia che si vuole implementare, in questo caso solo il metodo "compareTo".

Altro particolare da osservare nel seguente esempio sta nell'utilizzo di "int" e "num".

Il metodo "compareTo" associato alla classe Welcome ritorna un tipo "int", per quale motivo la variabile di destinazione è di tipo "num"?

Solitamente ci si aspetta che tipi come "int" e "double" siamo tipi primitivi, ma in realtà, in questo contesto, sono interfacce che estendono dell'interfaccia "num". Ciò significa che variabili di tipo "int" e "double" sono anche "num".

In Dart, spesso è possibile creare un oggetto a partire dalla sua interfaccia, senza la necessità di ricercare una classe specifica che implementa tale interfaccia.

Questo è possibile perché molte interfaccia hanno una classe **factory**, cioè una classe che crea oggetti che implementano l'interfaccia.

```
interface Person factory PersonFactory{
    Person(name);
    final name;
}
class PersonFactory{
    factory Person(name){
        if(name== null) return const Ghost();
            return new RealPerson(name);
    }
}
class RealPerson implements Person{
    RealPerson(this.name);
    final name;
}
class Ghost implements Person{
```

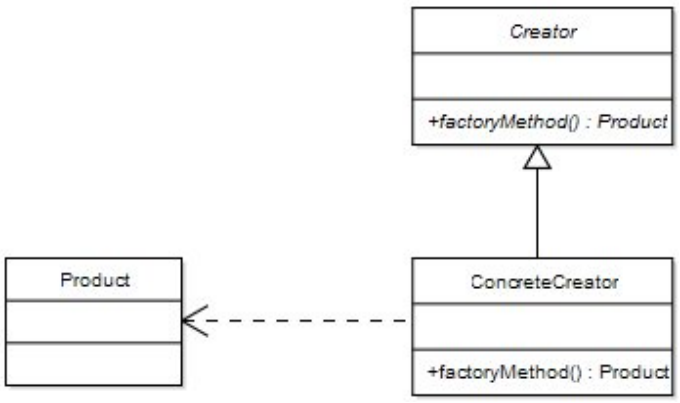

Figura 3.1: Struttura Factory Method

```
    const Ghost();
    get name() => "ghost";
}
main(){
    print(new Person("Pepita") is RealPerson);
    print(new Person(null) is Ghost);
}
```

Questo esempio mostra il funzionamento base delle *factory class*. Concettualmente alla base delle *factory class* c'è il **factory method**.

La creazione di un oggetto prevede la conoscenza del costruttore della classe. Il Factory fa parte di una famiglia di pattern detti *pattern creazionali*. Esso fornisce un'interfaccia comune per creare un oggetto, ma lascia alle sottoclassi il compito di decidere quale oggetto istanziare.(Vedi Figura 3.1)

"Creator" rappresenta l'interfaccia a cui ci si rivolge per la creazione degli oggetti, facendo un paragone con l'esempio fatto, rappresenta l'interfaccia "Person". Invece la classe "ConcreteCreator", che implementa l'interfaccia "Creator", è paragonabile alla classe "PersonFactory".

Le factory possono essere utilizzate in situazione in cui le istanze degli oggetti devono essere riciclate, oppure quando il codice di elaborazione deve essere separato dal codice di inizializzazione delle istanze delle classi.

3.2.1 Librerie

Le librerie sono una raccolta di risorse utilizzate per lo sviluppo software.

Lo scopo delle librerie è quello di fornire una collezione di entità di base pronte per l'uso, ovvero per il riuso di codice, evitando al programmatore di dover riscrivere ogni volta le stesse funzioni o strutture dati, facilitando così le operazioni di manutenzione.

Questa caratteristica si inserisce quindi nel più vasto contesto del "richiamo del codice" all'interno di programmi e applicazioni ed è presente in quasi tutti i linguaggi di programmazione.

I vantaggi principali derivanti dall'uso di un simile approccio sono i seguenti:

- è possibile separare la logica di programmazione di una certa applicazione da quella necessaria per la risoluzione di problemi specifici, quali calcolo di funzioni matematiche o la gestione di collezioni;
- le entità definite in una certa libreria possono essere riutilizzate da più applicazioni;
- è possibile modificare la libreria separatamente dal programma, senza limiti alla potenziale vastità di funzioni e strutture dati man mano disponibili nel tempo.

Quasi tutti i linguaggi di programmazione supportano il concetto di libreria e moltissimi includono delle librerie standardizzate, le cosiddette *librerie standard*. Si tratta di un insieme di funzioni e strutture dati che permettono di risolvere i problemi di programmazione più comuni, come ad esempio la libreria matematica o la libreria per le funzioni di I/O.

Dart permette agli sviluppatori di creare, utilizzando classi e interfacce, e utilizzare le librerie, le quali non cambiando durante l'esecuzione. Questo permette a porzioni di codice sviluppati individualmente di utilizzare librerie condivise.

Dart, oltre a permettere di sviluppare proprie librerie, fornisce le seguenti librerie per lo sviluppo web e server web:

- **Core Library**
  Libreria di base di Dart, contiene tutte le interfacce per il supporto

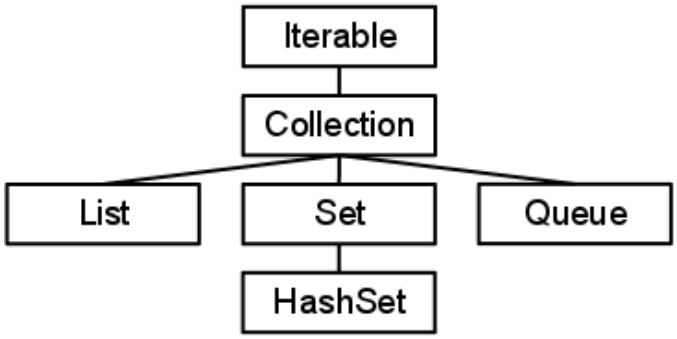

Figura 3.2: Iterable Hierarchy

delle strutture dati e per le relative operazioni.
Attualmente, la gerarchia delle interfacce per la Core Library è composta da tre parti principali, che estendono le seguenti interfacce:

- **Iterable** (Vedi Figura 3.2)
  L'interfaccia "Iterable" consente di ottenere un oggetto "Iterator". Tale interfaccia viene utilizzata dal costrutto "for-in" per "iterare" su un oggetto "Iterable".
  "Collection" è l'interfaccia che implementa "Iterable" e definisce le implementazioni dei metodi delle interfacce "Set", "List" e "Queue". Queste ultime rappresentano delle strutture dati.
  "HashSet" rappresenta un'implementazione più specifica dell'interfaccia "Set".
- **Map** (Vedi Figura 3.3)
  L'interfaccia "Map" rappresenta un contenitore di associazioni, mappando una chiave a un valore. I valori "Null" sono supportati.
  L'interfaccia "HashMap" è una versione più specifica dell'interfaccia "Map", essa non fornisce alcuna garanzia sull'ordine delle chiavi.
  L'interfaccia "LinkedHashMap" conserva l'ordine di inserimento.
- **Comparable, Hashable and Pattern** (Vedi Figura 3.3)
  L'interfaccia "Comparable" è implementata da "Data", "Duration", "Num" e "String".

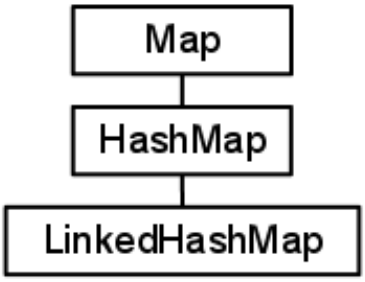

Figura 3.3: Map Hierarchy

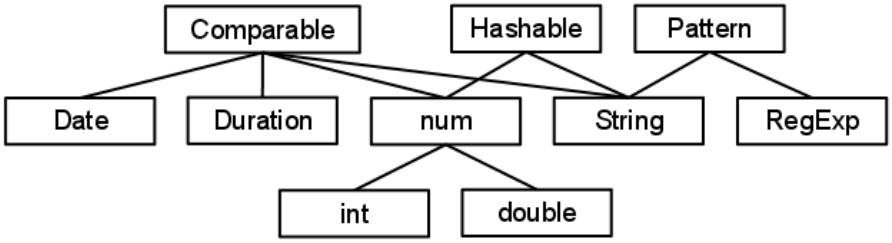

Figura 3.4: Comparable, Hashable and Pattern Hierarchy

L'interfaccia "Pattern" è implementata da "String" e "RegExp".

- **DOM Library**

  La libreria DOM contiene le interfacce per interagire con il DOM di Html5. Tale libreria è basata sullo standard Html5, cioè sulle specifiche del W3C/WHATWG.
  Tale interfacce evolvono in parallelo con gli standard Html5.

3.3 Tipi opzionali

Una delle caratteristiche più innovative di Dart è nell'utilizzo dei **tipi opzionali** (Optional Types).

Un tipo di dato è un "nome" che indica l'insieme di valori che una variabile, o il risultato di un'espressione, può assumere e le operazioni che su tali valori si possono effettuare. Dire per esempio che una variabile X è di tipo "integer" significa effermare che X può assumere come valori solo numeri interi che fanno parte di un determinato intervallo e che su tali valori sono ammesse solo certe operazioni, in questo caso, le operazioni aritmetiche.

Prendendo in considerazioni i tipi di dati, nell'insieme dei linguaggi di programmazione ve ne sono alcuni, come i linguaggi macchina della maggior parte degli elaboratori detti *linguaggio non tipizzati* in quanto non prevedono tipi di dati o consentono l'utilizzo dell'unico tipo, cioè una configurazione di bit, contenente tutti i valori possibili.

Oltre ai linguaggi non tipizzati, vi sono i *linguaggi tipizzati*. In questi linguaggi di programmazione è necessario associare alle variabili, o alle espressioni delle *annotazioni* o *dichiarazioni di tipo*.

In base al linguaggio di programmazione che si usa, queste annotazioni di tipo, devono essere specificate esplicitamente dal programmatore oppure possono essere generate in modo automatico dall'interprete o dal compilatore.

La modalità di assegnazione dei tipi alle variabili è detta *politica di tipizzazione*. Infatti, esistono due tipologie di linguaggi che si differenziano per la particolare politica di *tipizzazione* che utilizzano:

- **Linguaggi Statici** detti anche linguaggi a *tipizzazione statica*.

Questi linguaggi prevedono che il tipo di ogni variabile utilizzate all'interno del programma venga stabilito direttamente nel codice, dove viene assegnato esplicitamente per mezzo di parole chiave, come ad esempio int, long, float, char, ecc. Le variabili a cui vengono associati rigidamente dei tipi rimangono di quel tipo per tutto il programma.

I linguaggi a tipizzazione statica più comuni sono Java e C/C++.

- **Linguaggi Dinamici** detti anche linguaggi a *tipizzazione dinamica*.

  Questi linguaggi prevedono variabili con tipi ben definiti, ma a differenza dei linguaggi statici, tale tipo può cambiare dinamicamente durante l'esecuzione del programma, solitamente a causa di assegnazioni.

  I linguaggi a tipizzazione dinamica più comuni sono i linguaggi di scripting come JavaScript, cioè linguaggi interpretati, perché a causa della natura fortemente imprevedibile dei linguaggi a tipizzazione dinamica, l'interprete costituisce un ambiente di esecuzione "sicuro", in grado di assecondare tutti i cambiamenti di tipo delle variabili.

Per effettuare il controllo sui tipi di dati si ricorre al **type checking**.

Il type checking è un'operazione che consiste nel verificare e stabilire che i valori assegnati ad una variabile o le operazioni eseguite su una determinata variabile siano di un tipo ammissibile per il tipo della variabile, cioè permette di verificare se i vincoli imposti dai tipi sono soddisfatti.

Tale operazione è fondamentale per i linguaggi tipizzati, per poi non essere effettuata, invece, nei linguaggi non tipizzati.

Occorre però fare una distinzione, perché esistono due tipologie di controllo:

- **Type Checking Statico**

  Un linguaggio di programmazione utilizza un meccanismo di type checking statico se le operazioni di controllo sono eseguite solo a tempo di compilazione.

  Il type checking statico è inteso come una forma primitiva di verifica di un programma, infatti permette di individuare parecchi errori con largo anticipo, permettendo una migliore ottimizzazione del codice.

  Caratteristica molto importante è che il type checking statico permette la costruzione di codice che esegue molto velocemente.

- **Type Checking Dinamico**

  Un linguaggio di programmazione utilizza un meccanismo di type checking dinamico se le operazioni di controllo sono eseguite a tempo di esecuzione, cioè a run-time.

  Il type checking dinamico è più flessibile di quello statico, ma la flessibilità si "paga", infatti fornisce minori garanzie proprio perché non può operare a tempo di compilazione.

  Nel type checking dinamico i controlli si limitano alla correttezza sintattica degli operatori, ed eseguendoli a tempo di esecuzione sono, di solito, molto più sofisticati, e coinvolgono il processo di indentificazione di un tipo di una espressione.

  Oltre a tutto ciò, il type checking dinamico permette al compilatore di eseguire i suoi compiti molto più velocemente, dato che il controllo viene eseguito a run-time, permette l'uso di costrutti considerati illegali nei linguaggi statici, ed infine permette una più rapida prototipazione.

Dart è un linguaggio "dynamically typed", cioè è un linguaggio di programmazione a "tipizzazione dinamica", dove è possibile scrivere programmi che non utilizzano le annotazioni di tipo, proprio come si farebbe in JavaScript.

Dart, a differenza di JavaScript, nonostante sia a tipizzazione dinamica, ha la particolarità di poter scegliere se aggiungere o meno le annotazioni/dichiarazioni di tipo alle variabili del programma che si sta scrivendo.

L'aggiunta delle dichiarazioni di tipo, anche in caso di annotazioni incomplete o errate, non impedisce la compilazione o l'esecuzione del programma. Il programma avrà esattamente la stessa semantica, non importa la presenza o meno delle annotazioni di tipo.

Ecco un esempio di codice Dart non tipizzato.

```
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 esempio permette la creazione della classe "Point" composta dai parametri "x" e "y" e da due metodi "scale()" e "distance()".

Il prossimo esempio, a differenza di quello precedente utilizza un codice Dart tipizzato.

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

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

Questo esempio, semanticamente equivale all'esempio visto sopra, con l'aggiunta delle notazioni di tipo.

L'inserimento delle annotazioni di tipo all'interno del codice, cosa non possibile in JavaScript, offre dei vantaggi:

- È molto più facile leggere il codice con la presenza delle dichiarazioni di tipo;
- Consente di individuare facilmente in anticipo errori. Dart, infatti, consente un type checking statico il quale mette in guardia da eventuali errori, non ottenibili in altro modo.

Infatti, nonostante Dart sia un linguaggio dinamico, prevede, a scelta, un mix fra il type checking statico e il type cheking dinamico.

Il type checking statico permette di rilevare i potenziali problemi in fase di compilazione, e non in fase di esecuzione come in JavaScript.

La maggior parte dei "Warning" restituiti dal controllo sono legati ai tipi.

Il controllo statico non produce veri e propri "Error" sui tipi, tali da bloccare la compilazione e l'esecuzione, esso segnala "Error" nel momento in cui sono presenti problemi reali, in questo modo non costringe un'implementazione ristretta dovuta ai tipi del sistema.

Il type checking effettuato da Dart, a differenza di altri linguaggi, è più flessibile, segnala dei "Warning" dove altri avrebbero generato un "Error" bloccandone l'esecuzione a livello di compilazione. Dart non prevede un vero e proprio controllo rigido sui tipi, esso, infatti, non utilizza i tipi nello stesso modo dei sistemi classici.

```
String s1 = '9 ';
String s2 = '1 ';
int n = s1 + s2; // <--Warning
print(n);
// --> '9 1'
```

In questo caso, il controllo statico produce un "avviso", non blocca l'esecuzione del codice, infatti in output si avrà la stampa "9 1".

Quello che è stato appena mostrato è chiaramente un problema, tuttavia, a differenza di un sistema classico, un codice come il seguente

```
// funzione che ricerca all'interno di una tabella
Object lookup(String key){...}
String s = lookup('Test');
```

non causerà alcun "Warning" da parte del type checking, perché, in questo caso c'è una buona possibilità che il codice sia corretto, nonostante la mancanza di informazioni sul tipo.

Spesso l'implementatore ha una conoscenza semantica che il type checking non può avere, questo è uno dei motivi per cui il type checking utilizzato da Dart non è restrittivo come nella maggior parte del linguaggi statici.

Nel momento in cui non vengono forniti i tipi di dato, Dart come affronta eventuali "Warning" da parte del controllo?

La risposta va ricercata del tipo *Dynamic*, il quale rappresenta il "tipo predefinito" utilizzato quando non viene specificato nessun tipo di dato. L'utilizzo di questo tipo predefinito da parte di Dart, permette un controllo corretto da parte del type checking.

L'utilizzo o meno delle tipi è una scelta che, i creatori di Dart, hanno lasciato libera. Tutto dipende dalle abitudini di chi implementa. Non ci sono regole generali, anche se per esempio, si potrebbe sviluppare codice non tipizzato per la sperimentazioni di semplici prototipi di programmi, ma introdurre le annotazioni di tipo nel momento in cui l'applicazioni diventa più grande, in modo tale da facilitare il debugging e imporre una struttura più stabile all'applicazione.

3.4 Dart e Html

Dart è il nuovo linguaggio di programmazione struttura per il browser.

Come tutti i linguaggi orientati al web, come JavaScript, Dart ha la possibilità di essere direttamente integrato all'interno di pagine Html utilizzabili dal browser.

L'Html prevede un tag apposito per gli script.

```
<script>
    .....
</script>
```

Tale tag, può essere presente più volte all'interno della pagina, l'unica attenzione sta nel chiuderlo ogni volta che viene aperto.

Il tag di script Html fornisce una attributo "type", esso permette di definire la tipologia del linguaggio utilizzato per lo script.

Nel caso di JavaScript l'attributo di tipo va impostato con 'text/javascript'.

```
<script type='text/javascript'>
    .....
</script>
```

Invece per quanto riguarda Dart, l'attributo di tipo assume valore 'application/dart'.

```
<script type='application/dart'>
    .....
</script>
```

Equivalentemente al tag di script in JavaScript, in Dart, il contenuto dello script può essere in linea con il corpo del tag, oppure può essere specificato il riferimento URL al codice corrispondente, questo mediante l'utilizzo dell'attributo "src" del tag script.

Ogni script in Dart deve avere necessariamente la chiamata alla funzione *main()*, sia dichiarata direttamente nello script, oppure dichiarata all'interno dei file eventualmente importati. Questo perché nel momento in cui il browser effettua il caricamento di una pagina, quando incontra uno script Dart come prima cosa invocherà la funzione top-level (cioè di più alto livello) *main()*.

La funzione top-level *main()* rappresenta l'entry point di ogni script.

```
<script type='application/dart'>
    main(){
        ...
    }
</script>
```

All'interno di uno script Dart vi è la possibilità di importare risorse dall'esterno. Questo è permesso utilizzando #source oppure #import.

Gli script Dart permetto di includere script esterni utilizzando la direttiva #source, invece, per utilizzare librerie esterne occorre usare la direttiva #import.

Fin qui non è emerso niente di nuovo che già non veniva utilizzato in JavaScript. Invece, per certi versi, incorporare il codice Dart è diverso da incorporare JavaScript.

All'interno di ogni pagina Html possono essere presenti più tag di script, proprio come in JavaScript, ma in Dart, ogni tag di script nella stessa pagina viene eseguito in "isolamento".

Ciò significa che ogni singolo script ha il proprio contesto, in modo tale da non poter accedere al codice, quindi anche alle rispettive variabili dichiarate all'interno, di altri script presenti nella stessa pagina.

Dall'interno di uno script, l'unico modo per accedere a codice che non fa parte dello script stesso consiste nell'includere del codice dell'esterno, mediante l'utilizzo di #source e #import.

Quella appena descritta è una fondamentale differenza da JavaScript, anch'esso ha la possibilità di dichiarare più tag di script all'interno di un'unica pagina, ma a differenza di Dart, il contenuto degli script va a combinarsi all'interno di un'unico ambiente.

Ciò permette la visibilità del contesto tra i vari script presenti nella stessa pagina, con il rischio di creare conflitti sui nomi dichiarati, come per esempio variabili e funzioni.

Detto ciò, Dart essenzialmente richiede la presenza "obbligatoria" della funzione *main()* in ogni script definito all'interno di ogni pagina. Anche questa rappresenta una fondamentale differenza da JavaScript.

A differenza di JavaScript, Dart consente la dichiarazione di costrutti di alto livello come interfacce e classi, non sono presenti in JavaScript perché, come è stato detto in precedenza, esso è un linguaggio di programmazione object-based che non prevede l'utilizzo delle classi.

Ogni applicazione Dart, definita attraverso i tag di script, invece, prevede un esplicito entry point *main()* che viene invocato dal browser nel momento in cui esso è pronto per essere eseguito.

Di norma, ogni *main()* dello script Dart è invocato dall'evento "DOMContentLoaded". "DOMContentLoaded" è un evento, aggiunto alle specifiche dell'Html5, che viene invocato nel momento in cui tutti gli elementi che fanno parte del DOM della pagina sono stati caricati dal browser. Tale evento permette di differenziare il momenti in cui vengono caricati gli elementi essenziali della pagina da quegli elementi, come immagini e plugins, che impiegano molto tempo per essere caricati.

L'ordine di invocazione dei *main()*, da parte dell'evento "DOMContentLoaded" non è garantito.

Tutto ciò comporta alcune conseguenze:

- tutti i tags script con MIME type "application/dart" nella pagina possono essere caricati in modo asincrono/concorrente;
- il codice Dart viene eseguito solo dopo il controllo della pagina. Cioè, si controlla se la pagina è stata caricata tutta.

Tutto questo comporta un notevole ritardo nell'esecuzione del codice Dart.

Gli script in JavaScript, invece, vengono caricati "dall'alto verso il basso", cioè vengono caricati nell'ordine con cui sono stati dichiarati all'interno

della pagina html, ed ogni script, invece, a differenza di Dart, viene eseguito non appena conclude il caricamento. Ciò comporta un'attenzione maggiore su dove devono essere dichiarati gli script JavaScript, se nell'header o alla fine del body della pagina. Ma, in compenso, viene garantita maggiore velocità in fase di esecuzione.

Tale ritardo nell'esecuzione, da parte del codice Dart, sta portando gli sviluppatori del linguaggio, a considerare un modo per eseguire il codice Dart non appena avviene il caricamento. Ciò comporterebbe l'esecuzione di uno script prima che il DOM sia caricato completamente, sempre preservando allo stesso modo la semantica asincrona contemplata fin ad oggi.

Un'altra grande opportunità fornita da Dart, consiste nella possibilità di utilizzare un set di API più pulito da parte degli sviluppatori. Le API più importante utilizzate dai browser per la codifica sono le API DOM.

Le API DOM, utilizzate in JavaScript, stanno diventando sempre più complesse, tutto ciò è dovuto dai continui cambiamenti nel tempo e dall'introduzione di nuove tecnologia come XML.

Dart ha applicato semplici modifiche ai nomi delle elementi del DOM, per renderli più maneggevoli e facili da ricordare. Per fare un esempio, "HTMLElement" è diventato semplicemente "Element".

Il termine "HTML" è stato eliminato dalla maggior parte dei nomi dei tipi. Invece di "childNodes" e "children" ora si utilizza solo "nodes" e a sostituzione di "elements.ownerDocument" vi è "documents".

Gli sviluppatori di Dart hanno cercato di ottimizzare i nome degli elementi del DOM, per rendere più conciso e snello l'utilizzo di tali API.

Oltre a semplificare i nomi degli elementi del DOM, gli sviluppatori hanno cercato di semplificare quella parte di DOM rivolta alla ricerca degli elementi. Parte ancora oggi utilizzatissima da JavaScript, composta da una miriade di metodi fra i quali:

```
getElementByID()
getElementByTagName()
getElementByName()
getELementByClass()
querySelector()
querySelectorAll()
document.link
document.images
```

```
document.forms
document.scripts
formElement.element
selectElement.options
```

Tutta questa serie di metodi, è stata ridotta a soli due metodi:

- query()
- queryAll()

```
// JavaScript or Old Methods
elem.getElementById('foo');
elem.getElementByTagName('div');
elem.getElementByName('foo');
elem.getElementByClass('foo');
elem.querySelector('.foo .bar');
elem.querySelectorAll('.foo .bar');

//new Methods in Dart
elem.query('#foo');
elem.queryAll('div');
elem.queryAll('[name="foo"]');
elem.queryAll('.foo');
elem.query('.foo .bar');
elem.queryAll('.foo .bar');
```

L'idea di base consiste nell'uso di una libreria come quella di jQuery, ma senza avere a disposizione troppi e superflui metodi. Tutto ciò è stato pensato per rendere più snella e flessibile l'interfacciamento e la ricerca degli elementi DOM.

Anche i metodi utilizzati da JavaScript per manipolare gli attributi degli elementi sono stati modificati.

```
// JavaScript
elem.hasAttribute('name');
elem.getAttribute('name');
elem.setAttribute('name','value');
elem.removeAttribute('name');
```

```
//Dart
elem.attributes.contains('name');
elem.attributes['name'];
elem.attributes['name'] = 'value';
elem.attributes.remove('name');
```

A differenza di JavaScript, metodi come elements, nodes e query() ritornano degli oggetti, oggetti le cui interfacce sono implementate all'interno della librerie Dart.

Questi oggetti, che rappresentano gli elementi DOM, sono composti da strutture dati che rappresentano gli attributi degli elementi DOM. Questi attributi a loro volta sono caratterizzati da metodi per la manipolazione del contenuto.

Tutto ciò permette di rimuovere tutta una serie di metodi, utilizzati da JavaScript, associati agli elementi per la manipolazione degli attributi.

Oltre alla modifica delle API DOM per la ricerca e l'accesso agli elementi del DOM, sono state effettuate anche modifiche inerenti alla creazione degli elementi DOM.

Per la creazione di nuovi elementi, non è più necessario richiamare un metodo per la creazione, come viene fatto in JavaScript.

In Dart avendo a disposizione le classi caratterizzate da costruttori, per creare un nuovo elemento DOM occorre creare una nuova istanza di tipo DOM.

```
// JavaScript
document.createElement('div');

//Dart
new Element.tag('div');
```

L'idea di alleggerire e semplificare la libreria per accedere al DOM mostra una notevole differenza da JavaScript, caratterizzata da un'interfaccia complessa e piena di metodi, secondo il parere di Dart, superflui.

La nuova libreria per accedere al DOM, rende il codice più leggibile e interpretabile da parte di terzi, la quale rende il refactoring del codice più facile e veloce.

3.4.1 Eventi

La gestione degli eventi, rappresenta una degli argomenti su cui ruota la maggior parte delle caratteristiche che un'applicazione web deve avere.

Dart, proprio come JavaScript, gestisce l'interazione dell'utente con le applicazioni Web utilizzando il modello event-listener.

In entrambi i linguaggi, se si vuole gestire il verificarsi di un determinato evento, occorre effettuare la registrazione dell'event handler, cioè del gestore degli eventi.

JavaScript utilizza due metodi per registrare gli event handler:

- *metodo tradizionale* consiste nel associare direttamente un event handler sull'elemento che si vuole controllare;
- *metodo standard* rappresenta un metodo più moderno e funzionale, che associa l'event handler ad un determinato elemento attraverso addEventListener() e removeEventListener().

Il seguente esempio mostra come l'utilizzo del metodo tradizionale di JavaScript:

```
// JavaScript Tradizional Methods
elem.'onTypeEvent' = function(e){ ... };
```

Il metodo tradizionale utilizzato da JavaScript, effettua la registrazione del gestore degli eventi direttamente sulla proprietà "onTypeEvent" dell'elemento destinatario.

Uno degli svantaggi più considerevoli è che tale metodo non permette la registrazione di molteplici gestori degli eventi ma consente solamente l'associazione di un solo event handler per event.

Il metodo più utilizzato e più effiente per JavaScript è caratterizzato dal metodo standard:

```
// JavaScript
elem.addEventListener('typeEvent',
function(e){ ... }, false/true );
```

Il metodo standard utilizzato da JavaScript ha la caratteristica di essere soggetto ad errori dovuti all'individuazione dell'evento tramite l'uso di una stringa 'typeEvent'.

L'intento di Dart è quelli di rendere il processo di registrazione degli eventi semplice, rimuovendo eventuali errori da parte dell'implementatore.

Principalmente Dart ha rimosso completamente tutte le proprietà "onTypeEvent" da tutti gli elementi. In sostituzione ha creato la nuova classe "ElementEvents".

Ogni elemento Html all'interno di una pagina web è rappresentato da Dart dalla classe "Element".

Per gestire gli eventi, all'interno della classe "Element" è stata creata la classe "ElementEvents". La classe "ElementEvents" è possibile ottenerla utilizzando il metodo getter fornito dalla classe "Element".

```
// interfaccia Element
interface Element{
        ....
    ElementEvent get on();
        ....
}

// ottenere l'oggetto ElementEvent
elem.on();
```

All'interno della classe "ElementEvents" per ciascun tipo di evento noto, è presente una proprietà che lo caratterizza: click, mouseDown, ect.

```
// click event
elem.on.click
// focus event
elem.on.focus
//mouse event
elem.on.mouseDown
elem.on.mouseOver
elem.on.mouseOut
```

Tali esempi, mostrano alcune delle proprietà presenti all'interno della classe "ElementEvents".

Ognuna di queste proprietà rappresenta un oggetto evento che permette aggiungere o rimuovere gli ascoltatori e invocare gli eventi. La possibilità di aggiungere o rimuove eventi è data da due metodi: "add" e "remove".

```
// Dart
elem.on.click.add((event){
print('click');
})

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

I metodi "add" e "remove" hanno la seguente interfaccia

```
EventListenerList add(void handler(Event event),
                      [bool useCapture])

EventListenerList remove(void handler(Event event),
                         [bool useCapture])
```

Come viene mostrato dall'interfaccia, il parametro opzionale "useCapture" permette ad entrambi i metodi di abilitare o meno la fase di capturing, esattamente come era possibile in JavaScript.

Sostanzialmente Dart, decidendo di rimuovere le proprietà degli eventi dagli elementi, impedisce l'utilizzo del metodo tradizionale in uso in JavaScript, perché esso permetteva la registrazione di un solo event handler per evento.

Il metodo introdotto da Dart, sostanzialmente come il metodo standard di JavaScript, permette la registrazione di più event handler per evento, ma a differenza di JavaScript nasconde tutte le funzionalità di un evento dietro ad una singola proprietà.

3.5 Concorrenza

Il concetto di concorrenza è contrapposto a quello di sequenzialità.

In un sistema sequenziale i processi vengono eseguiti uno per volta e non si verifica alcuna forma di interazione tra essi durante l'esecuzione.

La concorrenza è una caratteristica dei sistemi nei quali può verificarsi che un insieme di processi siano in esecuzione nello stesso istante. L'esecuzione parallela può condurre a interazioni tra processi quando essi hanno a che fare con una risorsa condivisa.

La concorrenza può portare ad una serie di problematiche legate all'utilizzo di una stessa ricorsa condivisa da parte di più processi. Può causare:

- Corse Critiche (Race Conditions);
- Stallo (Deadlock);
- Starvation.

La programmazione concorrente, però, sfrutta alcuni principi per fronteggiare e risolvere questi tipi di problematiche.

Infatti, nell'ambito della comunicazione interprocesso (cioè fra processi) è di fondamentale importanza per gestire correttamente possibili situazioni di accesso a risorse condivise. I sistemi concorrenti mettono a disposizione delle primitive di comunicazione che permettono di gestire l'accesso ad una risorsa condivisa oltre a primitive di sincronizzazione che permettono in intervenire sulla sequenza secondo la quale avverranno determinati eventi. Quindi ogni sistema concorrente mette a disposizione le proprie primitive.

Dart e JavaScript sono entrambi linguaggi caratterizzati da un ambiente di programmazione a *single-threaded*, ciò significa che non è possibile mandare in esecuzione più script nello stesso momento.

Nonostante l'ambiente non sia multi-threaded, Dart fin da subito con l'utilizzo degli Isolate e JavaScript in seguito con l'introduzione da parte di Html5 dei WebWorker, permettono di supportare il concetto di concorrenza.

La maggior parte degli sviluppatori JavaScript, hanno tentato fino ad ora di "simulare" la concorrenza utilizzando tecniche come "setTimeOut()", "setInterval()", XMLHttpRequest e event handler.

Tutto queste tecniche si basano sull'esecuzione di operazioni in modo asincrono, cioè operazioni non bloccanti, ma ciò non implica necessariamente la concorrenza. Questo perché tutte le operazioni asincrone vengono elaborato dopo l'esecuzione dello script.

Le nuova specifica Html5 ha fornito a JavaScript nuove API, fra queste vi sono anche quelle per il supporto della concorrenza, cioè i WebWorker.

I WebWorker definiscono le API per la riproduzione in background degli script di una applicazione Web. Esso permettono di fare cose come mandare in esecuzione uno script che gestisce operazioni computazionali di alta

intensità, senza bloccare l'interfaccia grafica o gli script per la gestione delle interazioni con l'utente. Utilizzano la tecnica di "message passing" fra thread, per ottenere il parallelismo in esecuzione.

I WebWorker sono ideali per mantenere un'interfaccia utente sempre performante, aggiornata e reattiva agli utenti.

I WebWorker vengono mandati in esecuzione in un thread isolato. Quindi il codice che viene eseguito da un WebWorker deve essere contenuto in un file separato dal resto degli script.

```
//create the worker
var worker = new Worker('task.js');
```

Per poter utilizzare un WebWorker, occorre principalmente crearlo. Ciò avviene creando un nuovo oggetto "Worker" passando all'oggetto il riferimento al file contenente il codice da eseguire isolatamente. Se il file specificato esiste, il browser genera un nuovo "worker" thread, che scarica il file da eseguire in modo asincrono. Il "worker" non inizierà l'esecuzione del codice fino a quando il file non è stato completamente scaricato.

Dopo aver creato il "worker", è possibile mandarlo in esecuzione.

```
//star the worker
worker.postMessage();
```

La comunicazione fra un "worker" e la pagina principale avviene tramite il modello a *message passing*.

Il *message passing* è una forma di comunicazione utilizzata nella programmazione orientata agli oggetti e nella comunicazioni fra processi. In questo modello, i processi o gli oggetti sono in grado di inviare e ricevere messaggi (che comprende sequenze di byte, strutture dati oppure anche segmenti di codice) da altri processi. In sostanza, la comunicazione fra le parti, avviene solo e soltanto tramite lo scambio di messaggi.

Nel caso di JavaScript i "worker" comunicano con la pagina principale utilizzando il modello degli eventi e il metodo "postMessage()". A seconda del browser/versione che si utilizza, il metodo "postMessage()" può accettare sia stringhe che oggetti JSON come singolo argomento.

```
//main script
var worker = new Worker('doWork.js');
```

```
worker.addEventListener('message',function(){
    console.log('Worker said', e.data);
}, false);
worker.postMessage('Hello World');  // (1)

//doWork.js
self.addEventListener('message', function(e){ // (2)
    self.postMessage(e.data);
},false);
```

Questo esempio mostra come passare una stringa ad un "worker" e come esso restituisce semplicemente il messaggio passato.

Quando viene invocato il metodo "postMessage()" (1) dalla pagina principale, il "worker" gestisce il messaggio definendo un event handler (2) per l'evento "message". Il contenuto del messaggio passato è accessibile tramite la proprietà "Event.data".

L'esempio appena fatto, mostra che il metodo "postMessage()" rappresenta anche il mezzo attraverso il quale il "worker" passa i dati alla pagina principale.

I messaggi passati tra la pagina principale e il "worker", e viceversa, rappresentano una copia, non rappresentano una risorsa condivisa.

```
//main script
<button onclick="sayHI()">Say HI</button>
<button onclick="unknownCmd()">Send unknown command</button>
<button onclick="stop()">Stop worker</button>
<output id="result"></output>

<script>
       function sayHI() {
      worker.postMessage({'cmd': 'start', 'msg': 'Hi'});
    }

    function stop() {
        // Calling worker.terminate() from this
        // script would also stop the worker.
      worker.postMessage({'cmd': 'stop', 'msg': 'Bye'});
    }
```

```
    function unknownCmd() {
      worker.postMessage({'cmd': 'foobard', 'msg': '???'});
    }

    var worker = new Worker('doWork2.js');

    worker.addEventListener('message', function(e) {
      document.getElementById('result').textContent = e.data;
    }, false);
</script>

//doWork.js
self.addEventListener('message', function(e) {
  var data = e.data;
  switch (data.cmd) {
    case 'start':
      self.postMessage('WORKER STARTED: ' + data.msg);
      break;
    case 'stop':
      self.postMessage('WORKER STOPPED: '
        + data.msg + '. (buttons will no longer work)');
      self.close(); // Terminates the worker.
      break;
    default:
      self.postMessage('Unknown command: ' + data.msg);
  };
}, false);
```

Nel seguente esempio, a differenza del precedente, il messaggio passato al "worker" rappresenta un oggetto JSON. In tale esempio, la proprietà "msg" del messaggio JSON è accessibile sia dalla pagina principale che dal "worker". Infatti, sembra che l'oggetto JSON venga passato direttamente all'interno dello spazio di esecuzione del "worker", in realtà ciò che sta accadendo è che l'oggetto JSON viene serializzato e inviato al "worker", per poi essere successivamente deserializzato in ricezione sul "worker" stesso. La pagina principale e il "worker" non condividono la stessa istanza del messaggio.

Dall'esempio è possibile notare che vi è la possibilità di fermare l'esecuzione di un "Worker". Ciò è possibile invocando il metodo

```
worker.terminate();
```

dalla pagina principale, oppure

```
self.close():
```

all'interno dello stesso "worker".

Visto il modello di comunicazione utilizzato fra pagina principale e "worker", si suppone che l'ambiente del "worker" non è lo stesso della pagina principale. Nel contesto di un "worker", sia il riferimento "self" che "this" fanno riferimento all'ambito globale per il "worker".

Nonostante lo scope del "worker" non è lo stesso della pagina principale, un "worker" ha accesso ad un piccolo insieme di funzionalità di JavaScript, fra cui:

- l'oggetto "navigator";
- in sola lettura, l'oggetto "location";
- XMLHttpRequest;
- setTimerout() / clearTimeout() e setInterval() / clearInterval();
- l'Application Cache;
- l'importazione di script esterno con l'uso del metodo "importScripts()";
- la riproduzione di altri WebWorker.

Al contrario non ha accesso al DOM e agli oggetti "window", "document" e "parent".

Detto tutto ciò, ora ci si chiede, ma che genere di applicazione dovrebbe utilizzare i WebWorker?

Purtroppo, i WebWorker rappresentano una funzionalità relativamente nuova tale che la maggior parte di esempi/tutorial in rete li utilizzano per il calcolo dei numeri primi. Ecco alcune idee più interessanti su come impiegare le funzionalità dei WebWorker:

- ottenere e/o memorizzare dati per un uso successivo;
- controllo ortografico;
- analizzare dati audio e video;
- operazioni I/O in background o a polling su di webservices;
- elaborazione di array di grandi dimensioni;
- filtraggio di immagini in canvas;
- aggiornamento di un database in locale.

Anche Dart, proprio come JavaScript, è un linguaggio caratterizzato da un ambiente a single-theaded.

Proprio come JavaScript utilizza i WebWorker, introdotti dalla nuova specifica Html5, Dart, però utilizza gli Isolate per supportare il concetto di concorrenza.

Gli Isolate sono le unità di concorrenza utilizzate da Dart. Essi si basano sul modello ad attori.

Il modello ad attori è un modello di calcolo concorrente basato sugli attori, intesi come l'entità primaria.

Il modello ad attori adotta la filosofia che "ogni cosa è un attore". Tale filosofia è simile a "tutto è un oggetto", filosofia utilizzata dai linguaggi di programmazione orientata agli oggetti, la quale però differisce in quanto un software object-oriended solitamente viene eseguito in modo sequenziale, mentre il modello ad attori è intrinsecamente concorrente.

Dato che "ogni cosa è un attore", la comunicazione fra gli attori avviene utilizzando il modello message passing.

Un attore è un'entità computazionale che, in risposta ad un messaggio ricevuto, può contemporaneamente:

- inviare un numero finito di messaggi ad altri attori;
- creare un numero finito di attori;
- determinare il comportamento da utilizzare per il successivo messaggio che riceve.

All'interno di Dart l'unità di concorrenza è rappresentata dagli Isolate, che fondamentalmente rappresentano gli attori del modello sopra descritto.

Ogni Isolate è completamente indipendente l'uno dagli altri, ciò significa che ognuno di essi ha la propria memoria e il proprio flusso di controllo. Non vi è condivisione di risorse fra i vari Isolate, come per esempio variabili o metodi. L'unico meccanismo disponibile per far comunicare gli Isolate fra di loro è tramite lo scambio di messaggi. I messaggi vengono inviati attraverso le *ports*.

Le *ports* possono essere di due tipi:

- ReceivePort
- SendPort

Essi sono gli elementi che stanno alla base di un canale di comunicazione per gli Isolate. La prima classe, cioè ReceivePort, rappresenta il ricevitore, invece SendPort rappresenta il mittente all'interno del canale di comunicazione.

Ogni Isolate quando viene creato possiede come impostazione predefinita una porta di ricezione. Questa ReceivePort viene creata automaticamente e viene comunemente usata per stabilire la prima comunicazioni tra gli Isolate.

Dart possiede una libreria "dart:isolate" che definisce le API per la riproduzione e la comunicazione con gli Isolate.

```
interface Isolate{
    ...
    ReceivePort get port()
    ...
}

interface ReceivePort{
    ...
    void close()
    void receive(void callback(message, SendPort replyTo)
    SendPort toSendPort()
}
```

- close() chiude immediatamente la porta in ricezione. I messaggi in sospeso non vengono trattati ed è impossibile riaprire la porta.

- receive(...) imposta una funzione di callback in caso di ricezione di futuri o sospesi messaggi sulla porta di ricezione.
- toSendPort() consente di creare una nuova porta di trasmissione per l'invio di messaggi, tale porta è associata intrinsecamente alla corrispettiva porta di ricezione che l'ha creata.

```
interface SendPort{
    ...
    ReceivePort call(message);
    void send(message, [SendPort replyTo])
    ...
}
```

Le SendPort sono create dalle Receive Port. Ogni messaggio inviato attraverso una SendPort è consegnato alla sua rispettiva ReceivePort. Possono esistere molteplici SendPort per la stessa ReceivePort.

```
class HelloIsolate extends Isolate{
  HelloIsolate(){}
  void main(){
    print("HELLO before");
    port.receive((String message, SendPort replyTo){ // (1)
      print("HELLO ricezione");
      replyTo.send("Hello ${message}");
    });
    print("HELLO after");
  }
}

void main(){
  final ricezione = new ReceivePort();
  ricezione.receive((String message, SendPort notUsedHere){
    print("Messaggio Ricevuto: $message");
    ricezione.close();
  });

  HelloIsolate isol = new HelloIsolate();
```

```
  isol.spawn().then((SendPort sendPort){ // (2)
    sendPort.send("World", ricezione.toSendPort());
  });

  print("Fine Main!!");
}
```

Nel seguente esempio è stato creato un isolate dove sulla porta predefinita di ricezione è stata impostata una callback (1) che risponde con un saluto a tutti quelli che le inviano un messaggio.

Nel momento in cui l'Isolate viene creato, prima di poter inviare qualsiasi tipo di messaggio, occorre assicurarsi che l'isolate sia stato "caricato" completamente (2), questa operazioni è realizzata utilizzando "spawn().then( )".

Il risultato dell'esecuzione dell'esempio è il seguente:

```
// execution result
Fine Main!!
HELLO before
HELLO after
HELLO ricezione
Messaggio Ricevuto: Hello World
```

Per la creazione degli Isolate sono disponibile due tipologia di API: "spawnFunction" e "spawnUri".

```
SendPort spawnFunction(void topLevelFunction())
```

Consente la creazione di un Isolate, il quale utilizza il codice passato come argomento con codice del corrente Isolate.

```
SendPort spawnUri(String uri)
```

Crea e genera un isolate il cui codice è disponibile all'uri passato come parametro.

Entrambi creano un Isolate avente di default una ReceivePort, in più entrambe restituiscono una SendPort utilizzabile per l'invio dei messaggi all'Isolate.

A differenza dei WebWorker in JavaScript, gli Isolate non sono obbligatoriamente contenuti all'interno di un file esterno, ma possono essere presenti

all'interno del codice. Questo è possibile grazie alle classi che definiscono gli Isolate.

Data la novità degli argomenti, le API Dart che definiscono gli Isolate sono in continua evoluzione.

Momentaneamente sono presenti due tipologie di Isolate:

- Heavy Isolate
- Light Isolate

Gli Heavy Isolate, quando creati, vengono messi in esecuzione su un proprio thread privato. A differenza, i Light Isolate, quando sono in esecuzione vivono isolati all'interno dello stesso thread di colui che li ha creati.

I Light Isolate e differenza degli Heavy hanno la possibilità di accedere al struttura del DOM per la manipolazione dei contenuti.

Non si sa ancora per quanto tempo tale distinzione sarà presente.

Le API che definisco gli Isolate sono in continuo refactoring. Nuove API saranno prossimamente aggiunte, e altre saranno rimosse.

Nel prossimo futuro si sta già parlando di aggiunge un'API per creare DOM Isolate. Cioè Isolate con l'accesso al DOM, i quali, si prevede, saranno messi in esecuzione sul thread UI.

In ogni modo si spera che nel prossimo futuro, data l'importanza dell'argomento, Dart fornisca API ben definite per la creazione di applicazioni basate sulla concorrenza, in modo da rendere l'interazione con l'utente e le attività di computazione operazioni parallele.

Capitolo 4

Conclusioni

Negli ultimi tempi, l'utilizzo sempre più diffuso del modello Software as a Service, ha portato alla progettazione e realizzazione di applicazioni Web di ogni tipo.

Le nuove applicazioni Web, a differenza delle precedenti, ma ancora in uso, applicazioni proprietarie, sono accessibili tramite l'utilizzo di un semplice browser Web, non risiedono più all'interno degli elaboratori degli utenti, ma sono posizionate su server distribuiti all'interno della cosiddetta cloud (nuvola).

I vantaggi che derivano dall'uso di applicazioni Web sono molteplici. L'acquisto di una licenza non è più necessaria, basta effettuare un semplice abbonamento al servizio; non è più necessaria l'istallazione sul dispositivo e i dati risiedono tutti in remoto, di conseguenza l'accesso può essere effettuato da qualsiasi dispositivo.

Tutto ciò ha fatto aumentare sempre più la necessità e la richiesta di applicazioni Web.

L'obiettivo odierno del modello Software as a Service, consiste nel progettare e realizzare applicazioni Web che abbiano a disposizione tutte quelle funzionalità disponibili nelle applicazioni proprietarie. Si cerca perciò di fondere insieme i vantaggi introdotti dall'uso del cloud computing con le caratteristiche "classiche" da sempre in utilizzo sui software proprietari.

Per la realizzazione di queste applicazioni Web occorre l'utilizzo di linguaggi programmazione in grado di implementare applicazioni interattive e performanti allo stesso tempo.

JavaScript è un linguaggio di programmazione che fino a qualche anno fa, era utilizzato solo per la creazione di script che rendevano dinamica la pagina Web agli utenti. Ora dopo svariate modifiche ha assunto un ruolo fondamentale all'interno delle applicazioni Web. Esso rappresenta un mezzo di vera a propria programmazione software Web lato client, non più uno strumento per rendere solamente dinamica la pagina Web.

Alla fine del 2011, al centro del periodo nel quale le applicazioni Web stanno assumendo un ruolo fondamentale all'interno del Web, Google pubblica il suo nuovo linguaggio di programmazione: Dart.

Dart è un linguaggio di programmazione web strutturata, progettato per migliorare ed eventualmente sostituire JavaScript, o almeno per offrirne un'alternativa.

Dart e JavaScript, fondamentalmente hanno lo stesso obiettivo, cioè la programazione di applicazioni Web che rispettino il modello Software as a Service. Ora però, ci si chiede: ma Dart veramente offre tutte le funzionalità offerte da JavaScript? In quale modo Dart intende essere migliore di JavaScript?

A tali domande si è data una risposta facendo un'analisi accurata delle caratteristiche di entrambi i linguaggi.

Dart è un linguaggio di programmazione Web Class-based, cioè prevede l'utilizzo di classi e interfacce, a differenza di JavaScript, che non utilizza le classi, perché è un linguaggio di programmazione Object-based, cioè basato sugli oggetti.

Le classi e le interfacce forniscono un meccanismo ben compreso per definire in modo efficiente le API, questo perché esse rappresentano dei costrutti utilizzati come modelli per la creazione dei rispettivi oggetti. JavaScript non prevedendo l'utilizzo delle classi, non permette perciò la creazione di proprie API da utilizzare per eventuali altri progetti. Dart, quindi, per com'è strutturato il tipo di linguaggio consente la riusabilità del codice, caratteristica molto importante nella progettazione e sviluppo di software. In più questi costrutti consento poi l'incapsulamento e il riutilizzo di metodi e dati.

Nell'ingegneria del software, l'utilizzo delle classi è fondamentale. Nella progettazione di software svolge un ruolo importante l'utilizzo dell'UML, linguaggio di modellazione che permette di descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a tutti, per mezzo dell'uso del concetto di classe. In un linguaggio come Dart, che possiede il concetto di classe, il passaggio dalla fase di progettazione, effettuata per mezzo

di modelli UML, alla fase di implementazione risulta essere più efficiente e semplice, rispetto ad un linguaggio come JavaScript che non prevede il concetto di classe.

Anche la possibilità di realizzare il concetto di ereditarietà in Dart è molto più intuitivo rispetto all'uso dei prototipi in JavaScript. Esso non prevedendo il concetto di classe, l'unico meccanismo per simulare l'ereditarietà sta nell'utilizzo del prototipi poco intuitivi, ma soprattutto di difficile gestione nel momento in cui si ha a che fare con applicazioni di grandi dimensioni, con un numero elevato di tipi di oggetti.

Un'altra caratteristica che differenzia Dart da Javascript, sta nell'uso dei tipi. Entrambi sono linguaggi dinamici, cioè linguaggi a tipizzazione dinamica. Dart permette che ad ogni variabile possa essere o meno assegnato un tipo specifico. Ciò significa che se ad una variabile viene assegnato un certo tipo, questo non cambia per tutta l'esecuzione del programma e il controllo di tipo sull'utilizzo della variabile viene eseguito a livello di compilazione e non a livello di esecuzione come in JavaScript. Esso infatti non prevede l'assegnazione di tipo per alcuna variabile, ciò significa che in fase di esecuzione il tipo della variabile può cambiare in base alle operazioni che si effettuano, di conseguenza, il controllo sul tipo della variabile non può essere eseguito in fase di compilazione ma in fase di esecuzione.

Eseguire il controllo sui tipi delle variabili è un'operazione che, se fatta in fase di esecuzione, come nel caso di JavaScript, porta ad una esecuzione più lenta. Invece, se eseguita in fase di compilazione, come nel caso di Dart quando vengono effettuate le assegnazioni di tipo, per prima cosa permette un'esecuzione più veloce, e cosa più importante, permette il rivelamento di errori, dovuti ai tipi, in fase di compilazione, prima della messa in esecuzione. Ciò permette una maggiore padronanza di ciò che si è implementato.

Un'altro aspetto da non sottovalutare, riguarda la fase di progettazione. Nel caso si stia progettando un applicazione, soprattutto di grandi dimensioni, l'utilizzo dei tipi di dati sulle variabili permette una maggiore chiarezza nella lettura del codice, da parte di tutti coloro che partecipano alla progettazione o ne vogliano solo osservare il contenuto. In questo modo la manutenibilità e la riusabilità del codice da parte di terzi, non deve essere accompagna da commenti presenti all'interno del codice che "spiegano" come esso viene utilizzato, come viene fatto nella maggior parte dei programmi JavaScript, ma si utilizza intrinsecamente il codice stesso.

In entrambi i linguaggi vi è la possibilità di importare codice dall'esterno della pagina Web, ed all'interno di una pagina Html, i vari script possono essere dichiarati in qualunque punto si voglia, due aspetti importanti sono da tenere, però, in considerazione.

In JavaScript, nonostante gli script possano essere posti ovunque si voglia, occorre decidere meticolosamente su dove posizionarli. Questo perché, quando si effettua il caricamento della pagina, esso avviene "dall'alto verso il basso", e man mano che i vari script vengono caricati tra il codice html, essi vengono eseguiti. Questo non accade in Dart.

Ogni script Dart possiede un entry point che viene eseguito nel momento in cui tutta la pagina html è stata caricata. Ciò porta un notevole ritardo nell'esecuzione del codice rispetto a JavaScript, il quale esegue il codice ancora prima della fine del caricamento della pagina. Questo risulta essere una notevole limitazione per il linguaggio Dart, perché dovendo aspettare il caricamento di tutta la pagina html, se all'interno di essa vi sono elementi particolarmente "pesanti" da caricare, l'esecuzione degli script rischia di essere ritardata di molto, incorrendo nel rischio di avere un'interfaccia grafica bloccata. Tutto ciò va a favore di JavaScript.

Nello stesso tempo, JavaScript però deve fare i conti con l'ambiente di esecuzione degli script, perché i vari script che vengono dichiarati all'interno di una stessa pagina, non possiedono ognuno il proprio scope, come succede in Dart, ma fanno parte di uno scope unico dove si deve fare attenzione ai conflitti sui nomi delle variabili e delle funzioni. Questo, nell'ottica della progettazione di applicazioni di grandi dimensioni, porta a notevole problemi in fase in implementazione nel caso di un team di lavoro.

Per quanto riguarda gli eventi, fondamentalmente la gestione degli eventi non differisce fra i due linguaggi di programmazione. In entrambi, la gestione degli eventi avviene mediante la registrazione degli event handler sugli elementi destinatari. Ciò che Dart ha progettato è stato rimuovere dagli elementi la proprietà evento, ed aggiungere alla classe Element la proprietà ElementEvent. Essa racchiude in se e permette di gestire tutti gli eventi di quel determinato elemento. In più Dart impedisce l'utilizzo del metodo tradizionale utilizzato da JavaScript, perché tale metodo ha la limitazione di poter registrare un solo event handler per evento, invece Dart vuole lasciare il più libero possibile la registrazione degli event handler.

Grazie all'utilizzo delle classi, Dart ha la possibilità di racchiudere in un'unica proprietà tutti gli eventi su un determinato elemento. In Java-

Script, avendo solo a che fare con oggetti, tale cosa non sarebbe realizzabile. Oltre a questa particolarità, il funzionamento della gestione degli eventi fra i due linguaggi è sostanzialmente lo stesso.

Infine l'ultimo aspetto analizzato è quello della concorrenza. Dart e JavaScript sono entrambi linguaggi caratterizzati da un ambiente di programmazione a single-threaded, ciò significa che non è possibile mandare in esecuzione più script contemporaneamente. Nonostante ciò, Dart con l'utilizzo degli Isolate e JavaScript con l'utilizzo dei Web Worker, introdotti da Html5, permettono il supporto di una programmazione concorrente. Dart per supportare la concorrenza utilizza gli Isolate, essi sono creati a partire dalla classe Isolate, il codice eseguito da esso può essere interno o esterno alla pagina, a differenza dei WebWorker in JavaScript, che creano l'oggetto worker a partire da un solo file JavaScript esterno. Ciò permette a Dart una gestione molto più flessibile del codice di processi separati. In entrambi i casi, il metodo di comunicazione delle informazioni utilizzato consiste nello scambio di messaggi. JavaScript permette lo scambio di messaggi il cui contenuto è rappresentato da una stringa o da un oggetto JSON. Invece Dart permette lo scambio di qualsiasi tipo di oggetto. Come i WebWorker sono una novità introdotta dalla nuova specifica Html5, anche gli Isolate sono una tecnica ancora in fase di sviluppo, infatti le API fornite sono ancora in fase di modifica/sostituzione.

Dato un quadro generale delle tecnologie prese in considerazione, per quanto riguarda la progettazione e lo sviluppo di applicazioni Web, Dart essendo un linguaggio nato per la programmazione strutturata, possiede tutte quelle caratteristiche, come le classi e le interfacce, che stanno alla base di una buona progettazione software in campo di ingegneria del software.

Quindi Dart rappresenta un buon linguaggio di programmazione per le appliazioni Web di grandi dimensioni, le quali, proprio come le applicazioni di tipo proprietario, hanno alla base un lungo processo di analisi, progettazione e infine implementazione.

La possibilità di Dart di realizzare software riutilizzabile, manutenibile e scalabile rappresentano delle caratteristiche importanti per il linguaggio di programmazione.

Non costituisce nessun dilemma la questione di compatibilità con i browser, perché ad oggi i compilatori Dart trasformano il codice scritto in linguaggio Dart in sorgenti JavaScript, per tutti quei browser che non sono al momento compatibili.

A favore di JavaScript giocano i tanti anni di lavoro e di modifiche per consolidare tecniche già in utilizzo e per crearne di nuove in modo tale da adattare il linguaggio alle esigenze del Web. Negli anni i tanti programmatori JavaScript hanno acquisito una capacità tale di programmazione che gli risulterà difficile passare da un linguaggio come JavaScript a un linguaggio come Dart. Per chi però ha dimestichezza con la programmazione ad oggetti, e intende intraprende un percorso di formazioni per la programmazione del Web, naturalmente Dart non ha niente di meno di JavaScript e risulta uno strumento affidabile e potente.

Ringraziamenti

Inizio ringraziando di cuore le persone più vicine a me.

Ringrazio i miei amatissimi genitori Mauro e Flavia che mi hanno sempre sostenuto e supportata in tutte le mie scelte, mi hanno dato forza e coraggio per arrivare a dove sono oggi, sopportandomi giorno per giorno. Tutto ciò che sono oggi lo devo a loro e per questo non finirò mai di ringraziarli ed amarli.

Al mio fratellino Samuele che ha avuto tanta pazienza durante le lunghe sere in cui dovevo studiare e lui invece voleva che passassi un pò più di tempo con lui.

Alla mia nonna Giovanna che non si è mai scordata di un mio esame, mi ha sempre dato forza e il suo sostegno l'ha mostrato non dimenticandosi mai di accendere una candela ogni volta che avevo un esame per augurarmi buona fortuna.

Ai miei nonni Laura e Oreste che mi hanno sempre sostenuto e sono sempre stati disponibili a darmi ospitalità nei giorni in cui sono rimasta a dormire a Cesena.

Al mio fidanzato Francesco che mi è sempre stato affianco, ha sopportato i miei malumori e il mio "stress quotidiano" dovuto agli esami. Lo ringrazio perché nell'ultimo anno a causa degli ultimi esami l'ho trascurato tanto. Il suo amore e il suo sostegno mi ha sempre dato forza e coraggio per andare avanti.

Alla mia migliore amica di sempre Valentina, che da quando abbiamo iniziato l'università ci siamo viste sempre meno, ma non ci siamo mai dimenticate l'una dell'altra. La ringrazio per le lunghe chiacchierate al telefono, per i lunghi pomeriggi passati in biblioteca a studiare per gli ultimi esami, per aver sopportato le mie lamentele e per essermi sempre stata affianco.

Ci tengo a ringraziare Angelo e Pietro, compagni di università e amici sinceri. La loro compagnia ha reso le lezioni più leggere e diverten-

ti. Vi ringrazio perchè con voi questi anni di università sono stati unici e indimenticabili.

Al mio relatore, il Professore Alessandro Ricci vanno i miei sinceri ringraziamenti per la sua disponibilità, la fiducia e gentilezza mostratomi nella realizzazione di questa tesi.

Ci tengo anche a ringraziare il mio "cucciolo" Romeo, con la sua costante compagnia ha reso meno noiose tutte quelle giornate pessate chiuse in camera a studiare, regalandomi qualche sorriso fra una pagine e l'altra.

Infine voglio ringraziare tutti coloro che direttamente o indirettamente mi hanno sostenuto e hanno sempre creduto in me e nella strada che ho deciso di percorrere.

Bibliografia

[1] W3C HTML5. *http://www.w3.org/TR/html5/.*

[2] Wikipedia. *http://it.wikipedia.org*

[3] W3Schools.com - Html5. *http://www.w3schools.com/html5/*

[4] Html5 Rocks. *http://www.html5rocks.com*

[5] Html5 Today. *http://www.html5today.it*

[6] Html5 - La Guida Italiana. *http://www.guidahtml5.com/*

[7] Marijn Haverbeke: Eloquent JavaScript: A Modern Introduction to Programming, 2011

[8] David Flanagan: JavaSricpt The Definitive Guide, 6th Edition, O'Reilly, 2011

[9] John Resig: Pro JavaScript Techniques, Apress, 2006

[10] JavaScript.HTML.it. *http://javascript.html.it/*

[11] W3Schools.com - JavaScript. *http://www.w3schools.com/js/*

[12] Stacktrace. Aperiodico di resistenza informatic *http://stacktrace.it/tag/javascript/*

[13] Sviluppare in Rete *http://sviluppare-in-rete.blogspot.it/*

[14] DartLang *http://www.dartlang.org/*

[15] DartExperience *http://www.dartexperience.com*

[16] Seth Ladd's Blog *http://blog.sethladd.com/*

[17] Japh(r= By Chris Strom *http://japhr.blogspot.it/*

[18] Dart Programming Resources, Code and Community *http://dartr.com/*

[19] Dart News & Updates *http://news.dartlang.org/*

Elenco delle figure