Document Object Model

Luca Bompani

31 marzo 2000

Cos’è il DOM (1)

Il Document Object Model è un interfaccia di programmazione (API) per documenti HTML e XML. Definisce la struttura logica dei documenti ed il modo in cui si accede e si manipola un documento.

Utilizzando DOM i programmatori possono costruire documenti, navigare attraverso la loro struttura, e aggiungere, modificare o cancellare elementi.

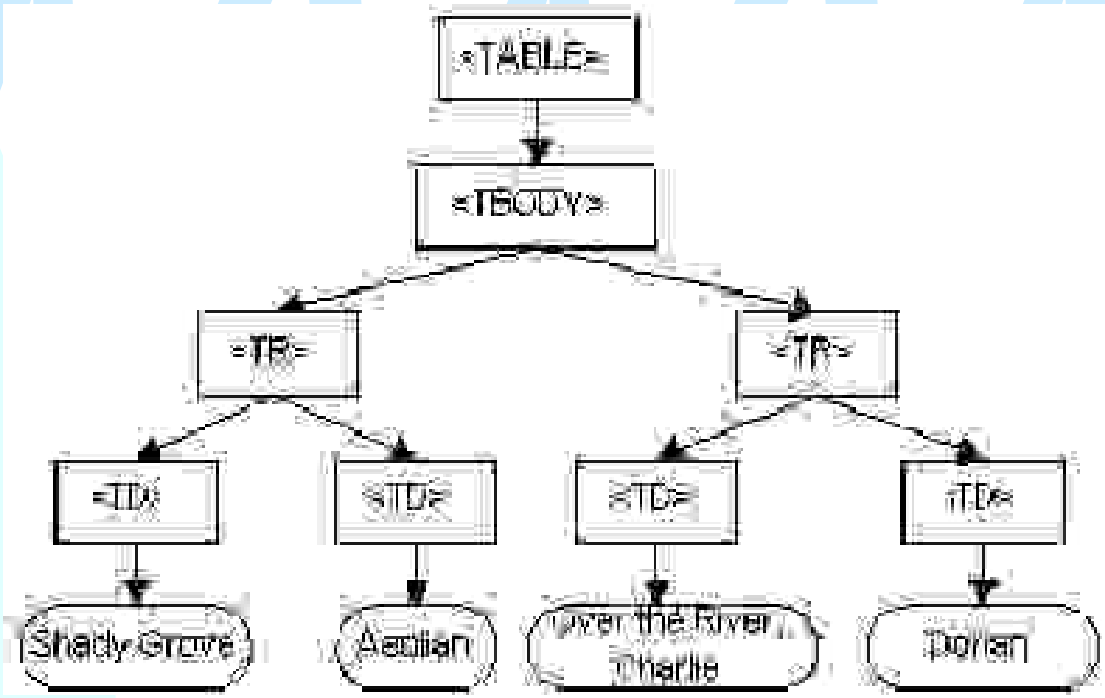
Ogni componente di un documento HTML o XML può essere letto, modificato, cancellato o aggiunto utilizzando l'Object Document Model.

Cos'è il DOM (2)

```
<TABLE>
  <TBODY>
    <TR>
      <TD>Shady Grove</TD>
      <TD>Aeolian</TD>
    </TR>
    <TR>
      <TD>Over the River, Charlie</TD>
      <TD>Dorian</TD>
    </TR>
  </TBODY>
</TABLE>
```


Cos'è il DOM (3)

Il Document Object Model rappresenta la tabella della slide precedente in questo modo:



Core (1)

Questa sezione dello standard DOM definisce un insieme di oggetti e interfacce per accedere e manipolare gli oggetti di un documento. Le funzionalità Core sono sufficienti per la creazione e manipolazione di documenti XML e HTML.

Core (2)

Le interfacce principali definite nella sezione Core sono:

- Node
- Document
- DocumentFragment
- DocumentType
- Entity Reference
- Element
- Attr
- ProcessingInstruction
- Comment
- CDATASection
- Entity
- Notation
- CharacterData

Node (1)

```
interface Node {
 const unsigned short ELEMENT_NODE = 1;
 const unsigned short ATTRIBUTE_NODE = 2;
 const unsigned short TEXT_NODE = 3;
 const unsigned short CDATA_SECTION_NODE = 4;
 const unsigned short ENTITY_REFERENCE_NODE = 5;
 const unsigned short ENTITY_NODE = 6;
 const unsigned short PROCESSING_INSTRUCTION_NODE = 7;
 const unsigned short COMMENT_NODE = 8;
 const unsigned short DOCUMENT_NODE = 9;
 const unsigned short DOCUMENT_TYPE_NODE = 10;
 const unsigned short DOCUMENT_FRAGMENT_NODE = 11;
 const unsigned short NOTATION_NODE = 12;
 DOMString nodeName;
 DOMString nodeValue;
 unsigned short nodeType;
 Node parentNode;
 NodeList childNodes;
```


Node (2)

```
  Node firstChild;
  Node lastChild;
  Node previousSibling;
  Node nextSibling;
  NamedNodeMap attributes;
  Document ownerDocument;
  Node insertBefore(Node newChild, Node refChild);
  Node replaceChild(Node newChild, Node oldChild);
  Node removeChild(Node oldChild);
  Node appendChild(Node newChild);
  boolean hasChildNodes();
  Node cloneNode(boolean deep);
  void normalize();
  boolean supports(DOMString feature,DOMString version);
  DOMString namespaceURI;
  DOMString prefix;
  DOMString localName;
};
```


Document (1)

```
Interface Document : Node {
 DocumentType doctype;
 DOMImplementation implementation;
 Element documentElement;
 Element createElement(DOMString tagName);
 DocumentFragment createDocumentFragment();
 Text createTextNode(DOMString data);
 Comment createComment(DOMString data);
 CDATASection createCDATASection(DOMString data);
 ProcessingInstruction createProcessingInstruction(
  DOMString target, DOMString data);
 Attr createAttribute(in DOMString name);
 EntityReference createEntityReference(DOMString name);
 NodeList getElementsByTagName(DOMString tagname);
```


Document (2)

```
 Node importNode(Node importedNode, boolean
deep);
 Element createElementNS(DOMString namespaceURI,
  DOMString qualifiedName);
 Attr createAttributeNS(DOMString namespaceURI,
  DOMString qualifiedName);
 NodeList getElementsByTagNameNS(DOMString
  namespaceURI, DOMString localName);
 Element getElementById(DOMString elementId);
};
```


CharacterData

```
interface CharacterData : Node {
 DOMString data;
 unsigned long length;
 DOMString substringData(unsigned long offset,
  unsigned long count);
 void appendData(in DOMString arg);
 void insertData(unsigned long offset,
  DOMString arg);
 void deleteData(unsigned long offset,
  unsigned long count);
 void replaceData(unsigned long offset,
  unsigned long count,DOMString arg);
};
```


Attr

```
Interface Attr : Node {
 DOMString name;
 boolean specified;
 DOMString value;
 Element ownerElement;
};
```


Element (1)

```
interface Element : Node {
 DOMString tagName;
 DOMString getAttribute(DOMString name);
 void setAttribute(DOMString name,DOMString value);
 void removeAttribute(DOMString name);
 Attr getAttributeNode(DOMString name);
 Attr setAttributeNode(Attr newAttr);
 Attr removeAttributeNode(Attr oldAttr);
 NodeList getElementsByTagName(DOMString name);
 DOMString getAttributeNS(DOMString namespaceURI,
  DOMString localName);
 void setAttributeNS(DOMString namespaceURI,
  DOMString qualifiedName, DOMString value);
 void removeAttributeNS(DOMString namespaceURI,
  DOMString localName);
```


HTML (1)

Questa sezione estende le interfacce Core, inserendo nuovi oggetti e proprietà specifiche dei documenti HTML.

Gli scopi di questa sezione sono:

- specializzare ed aggiungere funzionalità tipiche dei documenti HTML;
- fornire delle scorciatoie per le operazioni utilizzate più frequentemente in documenti HTML;
- garantire la compatibilità con i modelli implementati dai browser precedenti a DOM;

HTML (2)

La sezione HTML, in particolare, definisce queste interfacce:

- HTMLDocument: specifica le operazioni e le ricerche che possono essere effettuate su un documento HTML;
- HTMLElement: specifica le operazioni e le ricerche che possono essere effettuate su un generico elemento di un documento HTML;
- Specializzazioni di Element per tutti gli elementi HTML con attributi particolari;

HTMLDocument

```
interface HTMLDocument : Document {
 DOMString title;
 DOMString referrer;
 DOMString domain;
 DOMString URL;
 HTMLElement body;
 HTMLCollection images;
 HTMLCollection applets;
 HTMLCollection links;
 HTMLCollection forms;
 HTMLCollection anchors;
 DOMString cookie;
 void open();
 void close();
 void write(DOMString text);
 void writeln(DOMString text);
 NodeList getElementsByName(DOMString elementName);
};
```


HTMLElement

```
interface HTMLElement : Element {
 DOMString id;
 DOMString title;
 DOMString lang;
 DOMString dir;
 DOMString className;
};
```


HTMLLinkElement

```
interface HTMLLinkElement : HTMLElement {
 boolean disabled;
 DOMString charset;
 DOMString href;
 DOMString hreflang;
 DOMString media;
 DOMString rel;
 DOMString rev;
 DOMString target;
 DOMString type;
};
```


Views

Un documento può avere una o più viste associate, ad esempio una vista calcolata associate all'applicazione di uno stylesheet CSS. Una vista può essere statica, riflettendo cioè lo stato del documento nel momento in cui è stata creata, o dinamica, riflettendo le modifiche del documento nello stesso momento in cui avvengono. Questa sezione definisce l' interfaccia `AbstractView`, che introduce degli attributi in grado di mantenere il collegamento fra un documento e le sue viste.

AbstractView

```
interface AbstractView {
 DocumentView document;
};

interface DocumentView {
 AbstractView defaultView;
};
```

L'interfaccia `DocumentView` è implementata da tutti gli oggetti di tipo `Document`.

StyleSheets

L'interfaccia `StyleSheet`, definita in questa sezione, è un'interfaccia di base utilizzabile per rappresentare qualsiasi tipo di stylesheet.
Le implementazioni DOM che gestiscono stylesheet dovrebbero fornire una implementazione di questa interfaccia per ogni linguaggio di stile supportato.

StyleSheet

```
interface StyleSheet {
 DOMString type;
 boolean disabled;
 Node ownerNode;
 StyleSheet parentStyleSheet;
 DOMString href;
 DOMString title;
 MediaList media;
};
```


CSS

Le interfacce introdotte in questa sezione, sono state create per esporre le caratteristiche degli stylesheet CSS attraverso il modello DOM. L'interfaccia principale, `CSSStyleSheet`, è una specializzazione dell'interfaccia generica `StyleSheet`.

CSSStyleSheet

```
interface CSSStyleSheet : StyleSheet
{
 CSSRule ownerRule;
 CSSRuleList cssRules;
 unsigned long insertRule(DOMString rule,
   unsigned long index);
 void deleteRule(unsigned long index);
};
```


Events

La gestione degli eventi è stata introdotta all'interno del Document Object Model per raggiungere due obiettivi principali:

- definire un sistema ad eventi generico in grado di consentire la registrazione di event handler, controllare il flusso degli eventi e fornire informazioni sul contesto di ogni evento.
- definire un sottoinsieme comune fra i modelli ad eventi attualmente implementati dai browser, consentendo l'interoperabilità degli script.

EventTarget

```
interface EventTarget {
 void addEventListener(DOMString type,
  EventListener listener, boolean useCapture);
 void removeEventListener(DOMString type,
  EventListener listener, boolean useCapture);
 boolean dispatchEvent(Event evt);
};
```

Tutti i nodi DOM implementano questa interfaccia.

EventListener

```
interface EventListener {
 void handleEvent(in Event evt);
};
```

Questa interfaccia deve essere implementata da tutti gli event handler che sono interessati a ricevere un evento

Event

```
interface Event {
 const unsigned short CAPTURING_PHASE = 1;
 const unsigned short AT_TARGET = 2;
 const unsigned short BUBBLING_PHASE = 3;
 DOMString type;
 EventTarget target;
 Node currentNode;
 unsigned short eventPhase;
 boolean bubbles;
 boolean cancelable;
 void stopPropagation();
 void preventDefault();
 void initEvent(DOMString eventTypeArg,
  boolean canBubbleArg, boolean cancelableArg);
};
```


Traversal

Questa sezione introduce le interfacce `TreeWalker`, `NodeIterator`, e `Filter`. Il loro scopo è quello di fornire un semplice e robusto metodo per visitare ed attraversare i nodi di un documento DOM. `NodeIterator` presenta una vista piatta dei nodi di un sottoalbero, sotto forma di lista di elementi. Al contrario `TreeWalker` mantiene la struttura gerarchica del sottoalbero, consentendo di navigare attraverso le relazioni presenti fra gli elementi.

NodeIterator

```
interface NodeIterator {
 long whatToShow;
 NodeFilter filter;
 boolean expandEntityReferences;
 Node nextNode();
 Node previousNode();
 void detach();
};
```


TreeWalker

```
interface TreeWalker {
 long whatToShow;
 NodeFilter filter;
 boolean expandEntityReferences;
 Node currentNode;
 Node parentNode();
 Node firstChild();
 Node lastChild();
 Node previousSibling();
 Node nextSibling();
 Node previousNode();
 Node nextNode();
};
```


NodeFilter

```
interface NodeFilter {
 const short FILTER_ACCEPT = 1;
 const short FILTER_REJECT = 2;
 const short FILTER_SKIP = 3;
 const unsigned long SHOW_ALL = 0x0000FFFF;
 const unsigned long SHOW_ELEMENT = 0x00000001;
 const unsigned long SHOW_ATTRIBUTE = 0x00000002;
 const unsigned long SHOW_TEXT = 0x00000004;
 const unsigned long SHOW_CDATA_SECTION = 0x00000008;
 const unsigned long SHOW_ENTITY_REFERENCE = 0x00000010;
 const unsigned long SHOW_ENTITY = 0x00000020;
 const unsigned long SHOW_PROCESSING_INSTRUCTION =
  0x00000040;
 const unsigned long SHOW_COMMENT = 0x00000080;
 const unsigned long SHOW_DOCUMENT = 0x00000100;
 const unsigned long SHOW_DOCUMENT_TYPE = 0x00000200;
 const unsigned long SHOW_DOCUMENT_FRAGMENT =
  0x00000400;
 const unsigned long SHOW_NOTATION = 0x00000800;
 short acceptNode(Node n);
 };
```


Ranges

Un Range identifica un intervallo in un Documento.
è un intervallo continuo, nel senso che viene caratterizzato come il contenuto di tutti i punti compresi fra due estremi. Può ad esempio essere utilizzato da un text editor per evidenziare una certa area del documento.

Range (1)

```
interface Range {
 Node startContainer;
 long startOffset;
 Node endContainer;
 long endOffset;
 attribute boolean isCollapsed;
 Node commonAncestorContainer;
 void setStart(Node refNode, long offset);
 void setEnd(Node refNode, long offset);
 void setStartBefore(Node refNode);
 void setStartAfter(Node refNode);
 void setEndBefore(Node refNode);
void setEndAfter(Node refNode);
void collapse(boolean toStart);
```


Range (2)

```
 void selectNode(Node refNode);
 void selectNodeContents(Node refNode);
 short compareBoundaryPoints(CompareHow how,
  Range sourceRange);
 void deleteContents();
 DocumentFragment extractContents();
 DocumentFragment cloneContents();
 void insertNode(in Node newNode);
 void surroundContents(in Node newParent);
 Range cloneRange();
 DOMString toString();
 void detach();
};
```


Riferimenti

- ***Document Object Model (DOM)*** Level 1 Specification Version 1.0 W3C Recommendation 1 October, 1998, `http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001`

  Document Object Model (DOM) Level 2 Specification Version 1.0 W3C Candidate Recommendation 10 December, 1999, `http://www.w3.org/TR/1999/CR-DOM-Level-2-19991210`

  ECMAScript Language Specification Standard ECMA-262 2nd Edition - August 1998, `http://www.ecma.ch`