Seri Manajemen Proyek Software

Pemrograman JavaScript

Konsep dan Pembuatan Aplikasi menggunakan JavaScript

Last Updated:
5 Januari 2013

Pemrograman JavaScript

Deny Prasetyo Endy Muhardin

5 Januari 2013

Contents

1 Pendahuluan

1.1 Tentang JavaScript

1.2 Penggunaan JavaScript di masa kini

1.2.1 Penggunaan JavaScript di Browser

1.2.2 Penggunaan JavaScript di Mobile

1.2.3 Penggunaan JavaScript di Server

1.3 Tentang Buku JavaScript

1.3.1 Mengapa buku ini ditulis

1.3.2 Siapa yang sebaiknya membaca

1.3.3 Bagaimana urutan membacanya

1.3.4 Format penulisan

Agar lebih enak dibaca, kita akan membedakan bentuk dan warna tulisan sebagai berikut. Perintah yang kita berikan pada komputer ditulis seperti ini.

```
git --version
```

Hasil yang ditampilkan komputer ditulis seperti ini.

```
git version 1.7.4.1
```

Catatan khusus. Seringkali ada hal penting yang perlu mendapat perhatian khusus. Ini ditulis di menjorok ke tengah seperti contoh berikut.

> **Note**
>
> Working folder Git mengandung repository lengkap mulai dari revisi pertama sampai terbaru.

Berikut contoh kode program HTML.

```
<html>
    <head>
        <title>Halo Dunia</title>
    </head>
```

```
    <body>
        <h1>Halo Dunia</h1>
    </body>
</html>
```

Dan ini cara penulisan file konfigurasi

```
# Ignore file eclipse
.settings
.metadata
.project
.classpath
bin

# Ignore hasil kompilasi Maven
target
```

1.4 Lisensi

Buku ini memiliki lisensi Creative Commons Attribution Share Alike (CC-BY-SA). Artinya, semua orang:

- bebas menggunakan buku ini tanpa harus membayar, baik untuk keperluan non-profit maupun komersil. Anda boleh membuka pelatihan berbayar menggunakan buku ini.
- bebas membagikan buku ini kepada siapa saja.
- bebas membuat perubahan terhadap isi buku ini.

Semua kebebasan di atas hanya memiliki syarat yaitu tetap harus menyebutkan nama pengarang yang aslinya. Ini disebut dengan istilah attribution. Singkatnya, boleh dipakai dan dibagikan asal jangan diakui sebagai karya sendiri. Selain itu, segala perubahan yang dibuat juga harus dilisensikan sama dengan buku ini. Ini disebut dengan istilah Share-Alike. Lebih lanjut tentang lisensi ini bisa dilihat di website Creative Commons

1.5 Tools

Buku ini dibuat menggunakan perangkat pembantu :

- Markdown : format text untuk menulis buku
- Pandoc : aplikasi untuk mengkonversi markdown menjadi PDF atau HTML

1.6 Kontribusi

Semua orang boleh dan dianjurkan untuk ikut membantu penulisan buku ini. Bagaimana caranya? Gampang. Ada beberapa pekerjaan yang dapat dilakukan.

1.6.1 Reviewer

Tugasnya adalah memeriksa isi buku dan memberikan koreksi. Apa saja boleh dikoreksi, mulai dari tanda baca, salah ketik, contoh latihan tidak bisa dijalankan, apa saja. Kalau ada penjelasan yang kurang jelas juga boleh dikomentari. Apapun yang bisa membuat buku ini lebih baik. Hasil review dapat dientri di halaman Issue di Github.

1.6.2 Penulis

Bagus sekali kalau Anda ingin menyumbangkan tulisan. Lebih banyak yang mencerdaskan bangsa lebih baik. Begini caranya. Langsung saja fork repository `buku-js` ini dan segeralah berkarya. Begitu dirasa sudah memadai, kirimkan pull request ke saya. Nanti akan saya merge ke repository saya.

2 Persiapan

2.1 Daftar Tools yang dibutuhkan

2.1.1 Chrome Developer Tools

2.1.2 Firebug

2.1.3 Testacular

3 Konsep Dasar JavaScript

3.1 Primitive Data Type

3.2 Operators

3.3 Arrays

3.4 Conditional and Loop

4 Object Oriented JavaScript

4.1 Function and Closure

4.1.1 Callback Pattern

4.1.2 Returning Pattern

4.1.3 `this` Scope

4.1.4 Immediate Function

4.2 Object

4.3 Prototype

4.4 Inheritance

5 DOM Programming

5.1 Apa itu DOM? (sejarah-singkat-ngebut)

Document Object Model adalah API (Application Programming Interface) pada dokumen HTML dan XML. Representasi struktural pada dokumen yang memu-

ngkinan kita untuk memodifikasi konten dan penampilan visual (style document). Secara garis besar, DOM menjembatani interaksi pemrograman script dengan dokumen pada halaman web.

Dibawah komunitas internasional W3C yang dibentuk tahun 1994, Netscape, Microsoft dan perusahaan lain mengimplementasi standard baku pemrograman pada browser.

Versi awal "ECMAScript" dirilis pada 1997 menjadi standard untuk Javascript, JScript dan ActionScript. Tidak lama setelah itu, akhir 1998 W3C juga mulai mestandarkan DOM.

Tahap Awal standard DOM untuk web browser dikenal sebagai "DOM Level 1".

"DOM Level 2" diperkenalkan pada akhir tahun 2000. Mengusung *method* `getElementById` yang sangat terkenal dan juga *event model*.

"DOM Level 3" yang rilis sejak tahun 2004, masih menjadi kurikulum terkini bagi browser modern saat ini. Mengedepankan teknologi *XPath* dan *event handling* pada keyboard.

5.2 Javascript dan DOM

> Masihh binun, kk.., Lalu apakah DOM adalah bahasa pemrograman yang baru?

Jelas, bukan; tahan disitu. Bukan juga artinya yang sudah lama; DOM bukanlah bahasa pemrograman, melainkan suatu model. Tanpa DOM, pemrograman script kehilangan model untuk mendapatkan dan memanipulasi elemen pada halaman web.

Lebih lanjut interaksi javascript dan DOM akan kita uraikan dalam pembahasan Manipulasi DOM.

5.3 DOM Structure, InnerHTML, InnerText, Attribute, etc

5.3.1 DOM Structure

Setiap elemen pada dokumen web adalah suatu node yang terkait satu sama lain tergabung membentuk gugusan `object` dengan hirarki yang terstruktur seperti *pohon*.

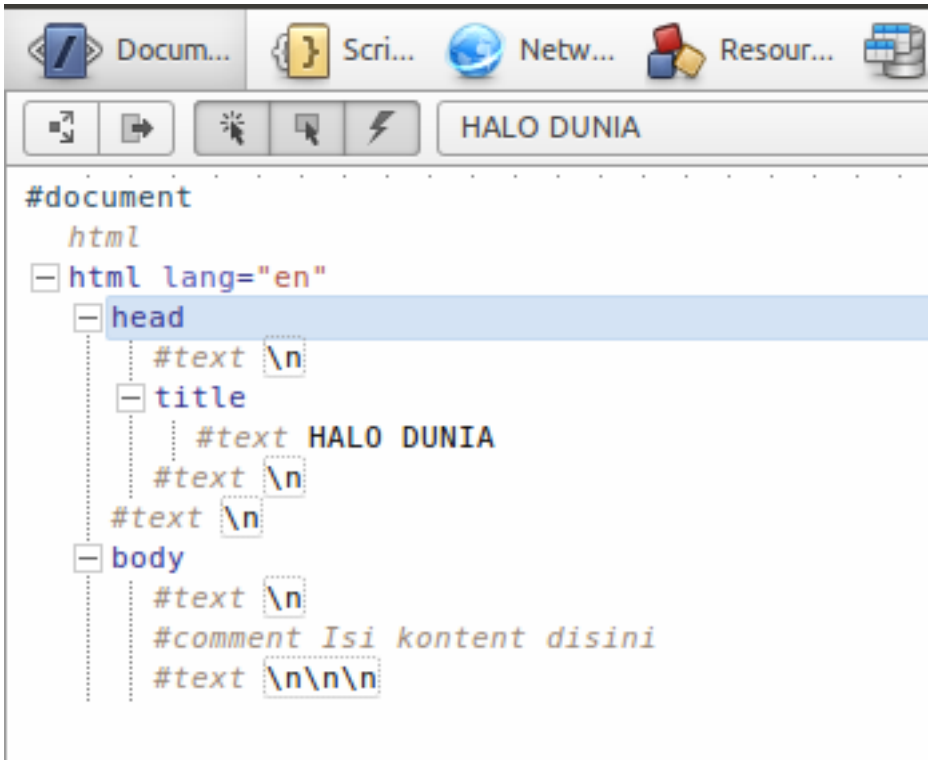
Ketika kita menulis suatu kode HTML, konten yang kita buat adalah suatu `Node` yang akan termuat pada suatu `Node` lainnya.

Indentasi markup bisa diterapkan pada penulisan kode HTML,. Hal ini mempermudah pembacaan kode dan tidak memberikan pengaruh yang berarti dari output hasil render browser.

Untuk mengetahui bagaimana browser melakukan proses parsing membentuk model DOM silahkan cekidot di mari, secara umum menjelaskan bagaimana markup HTML dibawah ini di terjemahkan:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HALO DUNIA</title>
</head>
<body>
<!-- Isi kontent disini-->
</body>
</html>
```

Saat kode HTML sederhana diatas diterima dan diproses oleh browser akan terbentuk sebuah dokumen berisi struktur `nodes` dalam format pohon. Dalam contoh ini, digunakan DOM Inspector dari browser Opera, Dragonfly.



Gambar 1

Seperti terlihat pada gambar, elemen-elemen pada dokumen yang terbentuk pada browser ditampilkan dalam struktur hirarki node dengan format pohon.

Node `document` disebut sebagai root yaitu *parent* teratas bagi semua elemen yang ada pada dokumen. Node `head` memiliki sebuah childNode yaitu `title`; Node `title` juga memiliki child yaitu node dengan tipe `Text` bernilai "*HALO DUNIA*", dst.

Object lain yang juga terlibat disini adalah DOM `window`. Object `window` merepresentasikan window itu sendiri. Object `window` dan `document` adalah variable global yang dapat diakses dari script pada dokumen.

Salah satu property dari `window` adalah `document` yang menunjuk kepada DOM `document` yang sedang diload pada window tersebut.

Lebih detail perhatikan potongan kode HTML berikut:

```
<a id="link-google" class="links" href="http://www.google.co.uk/">google</a>
```

Interface HTML diatas berupa tag **A**, menghasilkan sebuah link ketika diterima browser sebagai bagian dari sebuah dokumen.

Elemen node diatas memiliki sebuah child bertipe `Text` bernilai "google", juga dikenali memiliki beberapa attribute, diantaranya *id*, *class*, *href*.

5.3.2 Object `Node` types

Klasifikasi dari semua tipe node terdapat pada property object `Node`.

```
<script>
  for(tipe in Node)
    console.log(tipe + " = " + Node[tipe]);
/* # result from console #
ELEMENT_NODE = 1
ATTRIBUTE_NODE = 2
TEXT_NODE = 3
CDATA_SECTION_NODE = 4
ENTITY_REFERENCE_NODE = 5
ENTITY_NODE = 6
PROCESSING_INSTRUCTION_NODE = 7
COMMENT_NODE = 8
DOCUMENT_NODE = 9
DOCUMENT_TYPE_NODE = 10
DOCUMENT_FRAGMENT_NODE = 11
NOTATION_NODE = 12
DOCUMENT_POSITION_DISCONNECTED = 1
DOCUMENT_POSITION_PRECEDING = 2
DOCUMENT_POSITION_FOLLOWING = 4
DOCUMENT_POSITION_CONTAINS = 8
```

```
DOCUMENT_POSITION_CONTAINED_BY = 16
DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC = 32
toString = function toString() { [native code] }
*/
</script>
```

Beberapa tipe node yang akan sering kira temui yaitu tipe ELEMENT_NODE dan TEXT_NODE. Konstanta bernilai integer ini dapat digunakan untuk mengindentifikasi tipe node elemen pada dokumen dengan mengembalikannya dari property `nodeType`.

Kita dapat perlihatkan bahwa `object document` adalah Node dengan tipe `DOCUMENT_NODE`.

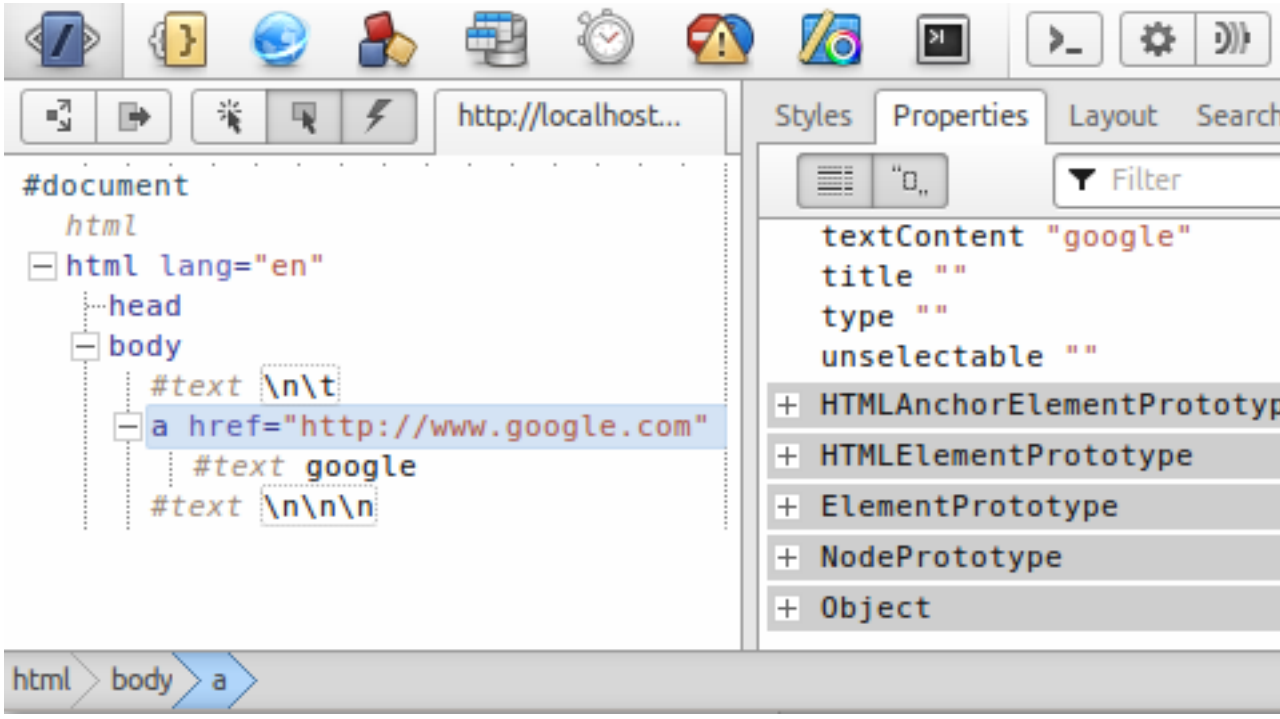
```
<script>
  console.log(document.nodeType == Node.DOCUMENT_NODE);
</script>
```

Setiap node yang ada pada dokumen, termasuk dokumen itu sendiri memperoleh sifat dasar turunan dari object `Node`. Property dan method yang diturunkan tergantung dari tipe elemen.

Berikut ini pewarisan dari beberapa node:

- `Object` < `Node` < `Element` < `HTMLElement` < `HTML[*]Element`
- `Object` < `Node` < `CharacterData` < `Text`
- `Object` < `Node` < `Document` < `HTMLDocument`

```
<a href="http://www.google.com/">Google</a>
```



Gambar 2

Pada contoh diatas kita miliki suatu link tag **A**. Melalui DOM inspector elemen node ini adalah turunan dari `HTMLAnchorElement`, `HTMLElement`, `Element`, `Node`, `Object`.

5.3.3 Properties and Method of `Node`

Seperti telah disebutkan, bahwa elemen-elemen pada dokumen adalah suatu `Node` berikut *property* dan *method* yang juga diturunkan dari object `Node` tersebut. Property dan method ini menjadi nilai dan fungsi dasar yang relevan bagi tiap elemen dan dapat digunakan untuk manipulasi dan melakukan DOM programing.

Sub bab ini tidak akan menjelaskan semua *method* dan *properties*. Penjelasan serta penggunaan akan disinggung pada pembahasan selanjutnya, lagi-lagi setelah kita mendefiniskan suatu node.

Beberapa property dan method dari elemen Node yang umum digunakan.

Node properties

- element.`nodeValue`
- element.`nodeName`

- element.`innerHTML`
- element.`parentNode`
- element.`childNodes`
- element.`firstChild`
- element.`lastChild`
- element.`nextSibling`
- element.`previousSibling`

Node Method:

- element.`appendChild()`
- element.`insertBefore()`
- element.`cloneNode()`
- element.`removeChild()`
- element.`replaceChild()`
- element.`setAttribute()`
- element.`getAttribute()`
- element.`addEventListener()`

Document Method:

- `document.getElementById()`
- `document.getElementsByTagName()`
- `document.getElementsByClassName()`
- `document.createElement()`

Window Properties:

- `window.location`
- `window.navigator`
- `window.localStorage`

Window Method:

- `window.scrollTo()`

Beberapa catatan mengenai DOM:

- DOM merupakan hirarki terstruktur dari konten dokumen yang terdiri dari kumpulan nodes. Beberapa tipe DOM nodes diantaranya: `Element`, `Text`, `Document`.
  - Tipe node `Element` adalah setiap elemen yang terdapat pada dokumen. Misalnya suatu link pada halaman web, adalah elemen node dengan *interface* HTML '<a>'
  - Tipe node `Text` adalah text yang termuat pada node `Element` pada dokumen.
  - Tipe `Document` adalah dokumen itu sendiri. Ini adalah akar node (root) dari hirarki DOM / struktur pohon.
- Setiap Node DOM adalah suatu *object*.
- Object `window` bersifat sebagai *global object*, dan dapat diakses melalui script sebagai `window` dan selayaknya object, ia memiliki *property* dan *method*.

5.4 Finding Element

Terdapat beberapa metode untuk memperoleh elemen node pada dokumen. Satu hal yang perlu dicermati adalah hasil kembalian dari metode yang digunakan.

Single Node

Metode umum yang sering digunakan untuk memperoleh elemen node tunggal:

- `querySelector()`
- `getElementById()`

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HALO DUNIA</title>
</head>
<body>
```

```
  <p>Selamat datang</p>
  <a href="http://www.google.com">google</a>
  <p id="penutup">selamat tinggal</p>
<script>
  console.log(document.querySelector("p").textContent)
  console.log(document.getElementById("penutup").textContent)
</script>
</body>
</html>
```

Method `querySelector()` mengembalikan node pertama yang cocok dengan query yang diberikan. Kelebihan yang dimiliki oleh method ini adalah kita dapat menggunakan format standar seperti pada selector CSS 3. Sehingga kita dapat memperoleh elemen seperti ini:

```
  console.log(document.querySelector("p:last-child").textContent)
```

Lain halnya dengan `getElementById()`, method ini mencari elemen dengan attribut *id* spesifik yang ada pada suatu elemen. Idealnya attribut *id* bagi elemen adalah unik, namun apa yang terjadi jika secara tidak sengaja ataupun disengaja nilai *id* yang sama diberikan pada elemen berbeda? Dalam hal pencarian node dengan menggunakan `getElementById()`, maka node pertamalah yang akan dikembalikan.

Mungkin ada pertanyaan yang sedikit mengusik, mengapa `document`? Bagaimana jika...? Mungkinkah bila...? Okay mari kita kupas sedikit lebih rinci.

Kita ketahui bahwa `document` adalah suatu `Node`, lebih spesifik boleh dikatakan `document` node merupakan root atau parent bagi semua elemen yang ada di dalam dokumen. Penetapan node parent seperti kita menetapkan lingkup daerah pencarian. Dan seperti dijanjikan bahwa setiap elemen memiliki method dan property yang diturunkan dari `object Node`, maka

```
<!DOCTYPE html>
<html lang="en">
<body>
  <p>Selamat datang</p>
  <div id="kakikaku">
     <p>selamat tinggal</p>
  </div>
<script>
  // definisikan variable parent sebagai suatu node
  var parent = document.querySelector("div");
  console.log(parent.querySelector("p").textContent); // "selamat tinggal"
</script>
</body>
</html>
```

Pencarian node `querySelector()` dengan query "p" diatas kita persempit pada parent node dengan *interface* HTML "div" pertama, sehingga kembalian node pertama yang sesuai untuk query ini memiliki property `textContent` bernilai "*selamat tinggal*", berbeda dengan yang dihasilkan oleh kode dibawah ini:

```
console.log(document.querySelector("p").textContent);
```

Bagaimana dengan method `getElementById()`? Sayangnya method ini tidak diturunkan untuk semua elemen, melainkan hanya node `document`. Hal ini relevan dengan keadaan ideal yang diharapkan dalam pemberian attribute *id* unik untuk pencarian elemen dengan dengan method ini.

Namun jangan kuatir masih ada method turunan lain yang dinilai cukup powerfull dalam pencarian node.

NodeList

Metode untuk pencarian node dengan kembalian berupa koleksi node:

- `querySelectorAll()`
- `getElementsByTagName()`
- `getElementsByClassName()`

Nilai yang dikembalikan dari method-method ini adalah berupa `NodeList`.

Mengurai elemen yang dikembalikan berupa `NodeList` mirip seperti kita lakukan pada elemen-elemen dalam suatu `Array`, namun perlu diingat bahwa `NodeList` BUKAN sebuah `Array`. Sehingga method yang terdapat pada `Array` (seperti: forEach, map) tidak bisa diterapkan pada `NodeList`.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HALO DUNIA</title>
</head>
<body>
  <p>Selamat datang</p>
  <a href="http://www.google.com">google</a>
  <p id="penutup">selamat tinggal</p>
<script>
  var paragraf = document.querySelectorAll("p");
  console.log(Array.isArray(paragraf)); // false
  console.log(paragraf[0].textContent);
</script>
</body>
</html>
```

Kabar baiknya disini, method dengan kembalian berupa `NodeList` seperti diatas diwarisi untuk setiap elemen node. Sehingga pencarian spesifik dibawah parent node tertentu memungkinkan untuk dilakukan.

XPathResult

XPath bukanlah method dari constructor `Node`, namun kita akan sedikit bahas bagaimana kita bisa mencari elemen node menggunakan teknik ini.

Jika `querySelector()` menggunakan query String, pada XPath istilah yang digunakan adalah ekspresi XPath String dengan standar format penulisan yang sedikit berbeda.

XPath (dari akronim XML Path Language) menggunakan sintaks non-XML yang sangat fleksibel menunjuk suatu atau beberapa node pada dokumen berbasis XML. Teknik pencarian dengan XPath merupakan alternatif yang cukup handal menavigasi elemen pada DOM dokumen selain fitur standar yang ada pada DOM Core.

Untuk mengeksekusi XPath memerlukan method `document` yaitu `evaluate()`. Kembalian method ini adalah berupa `object XPathResult`, dan kembalian ini bergantung dari parameter `resultType` yang diberikan.

```
<script>
var xpathResult = document.evaluate(
 xpathString,
 contextNode,
 namespaceResolver,
 resultType,
 result
);
</script>
```

- `xpathString` string mewakili ekspresi xpath yang akan di evaluate
- `contextNode` node yang menentukan parent konteks pencarian. Jika tidak spesifik diberikan, `contextNode` umumnya bernilai `document`.
- `namespaceResolver` fungsi yang akan dilewatkan prefix namespace dan mengembalikan URI namespace yang diasosiasikan dengan prefix yang diberikan tersebut. `null` adalah nilai yang umum diberikan untuk dokumen HTML atau ketika tidak ada namespace spesifik yang diberikan.
- `resultType` integer mewakili tipe hasil yang akan dikembalikan. Beberapa konstan dari constructor `XPathResult` bernilai integer dari 0 s/d 9. Lebih lengkapnya silahkan cekidot named result constant Akses node yang umum digunakan bisa menggunakan tipe Result `NODE_SNAPSHOT` yang memungkinkan memodifikasi node lebih lanjut pada dokumen.

```
<!DOCTYPE html>
<html lang="en">
<body>
  <div>Pada suatu hari...</div>
  <div id="kakikaku">
     <p>selamat tinggal</p>
  </div>
<script>
var nodes = document.evaluate("/html/body//div",
  document,
  null,
  XPathResult.ORDERED_NODE_SNAPSHOT_TYPE,
  null);
/*
* cari pada parent node document, semua elemen DIV.
* hasil akan berupa snapshot koleksi node terurut
*/
var nodeLen = nodes.snapshotLength;
console.log(nodeLen);
for(var i=0; i<nodeLen; i++){
  var node = nodes.snapshotItem(i);
  console.log(node.innerHTML)
}
</script>
</body>
</html>
```

Perlu diketahui `resultType` yang diberikan berupa `ORDERED_NODE_SNAPSHOT_TYPE`, variable `nodes` diatas bukanlah berupa Array melainkan koleksi snapshot (`object XPathResult`). Mengurai elemen pada hasil `XPathResult` dengan `resultType` ini menggunakan method `snapshotItem`.

Lebih ringkas, penulisan ekspresi xpath diatas dapat ditulis seperti dibawah ini:

```
<script>
var nodes = document.evaluate("//div",
  document,
  null,
  XPathResult.ORDERED_NODE_SNAPSHOT_TYPE,
  null);
</script>
```

Tipe `resultType` lain bisa juga menggunakan `UNORDERED_NODE_ITERATOR_TYPE` atau `ORDERED_NODE_ITERATOR_TYPE`. Kembalian juga berupa koleksi nodes `XPathResult`, namun untuk mengurai setiap node pada hasil menggunakan metode `iterateNext`.

perhatikan potongan kode-script berikut:

```
<script>
nodes = document.evaluate("//div",
  document,
  null,
  XPathResult.ORDERED_NODE_ITERATOR_TYPE,
  null);
/*
* cari pada parent node document, semua elemen DIV.
* hasil akan berupa iterator node terurut
*/
while(node = nodes.iterateNext()){
  console.log(node.innerHTML);
}
</script>
```

Perbedaan yang perlu diketahui disini hasil `XPathResult` dengan `resultType` berupa `ORDERED_NODE_ITERATOR_TYPE` tidak memiliki nilai property `snapshotLength`. Selain itu, sejak hasil ini dikembalikan apabila terjadi modifikasi pada dokumen, koleksi node pada hasil tersebut tidak lagi valid untuk diakses, (Exceptions raised).

perhatikan potongan kode-script berikut:

```
<script>
var nodes = document.evaluate("//div",
  document,
  null,
  XPathResult.FIRST_ORDERED_NODE_TYPE,
  null);
/*
* cari pada parent node document, elemen pertama DIV.
* hasil akan berupa snapshot node terurut
*/
var node = nodes.singleNodeValue;
console.log(node.innerHTML);
</script>
```

Pada potongan kode diatas diberikan `resultType` berupa `FIRST_ORDERED_NODE_TYPE`, hasil yang dikembalikan ke nodes memungkinkan kita memiliki property `singleNodeValue` dari constructor `XPathResult`. Nilai ini adalah node pertama yang sesuai dengan pencarian xpath string.

Penggunaan ekspresi path lebih lanjut bisa juga kita melakukannya seperti dibawah ini:

```
<script>
var nodes = document.evaluate("//div[@class=\"konten\"]",
  document,
  null,
  XPathResult.ORDERED_NODE_SNAPSHOT_TYPE,
  null);
/*
* cari pada node document, semua elemen DIV
* yang memiliki attribute class "konten".
* hasil akan berupa snapshot koleksi node terurut
*/

</script>
```

Atau cara lain seperti potongan kode berikut:

```
<script>
var nodes = document.evaluate("//a[contains(@href, \"google\")]",
  document,
  null,
  XPathResult.ORDERED_NODE_SNAPSHOT_TYPE,
  null);
/*
* cari pada node document, semua elemen A
* yang memiliki attribute href
* yang mengandung potongan string "google"
* hasil akan berupa snapshot koleksi node terurut
*/

</script>
```

Walau terlihat lebih rumit dari teknik yang ada pada DOM Core, teknik pencarian dengan XPath ini setidaknya memberikan secercah harapan dan keleluasaan dalam mencari dan menggali node elemen pada DOM dokumen.

5.5 Manipulasi DOM

Berbekal pendefinisian node mengunakan beberapa cara dalam pencarian elemen node, kita akan sundul kembali properties dan method dari Node.

Berbagai hal yang memungkinkan kita melakukan perubahan pada DOM atau sekedar mendapatkan nilai tertentu, bahkan menghapus suatu node pada DOM.

5.5.1 Mendapatkan dan Menetapkan Attribute elemen node

Attribute pada elemen memiliki arti dan tujuan tertentu yang memberikan efek secara langsung atau tidak langsung bagi elemen itu sendiri.

`Object Node` memiliki method `getAttribute()` dan `setAttribute()` untuk memberikan atau mendapatkan suatu attribute pada elemen node.

```
<!DOCTYPE html>
<html lang="en">
<body>
  <a title="Pergi ke bulan" href="http://www.google.co.uk">Google</a>
<script>
  var el = document.querySelector("a");
  console.log(el.getAttribute("title"));

  // modifying
  el.setAttribute("title", "Pergi ke Google");
  console.log(el.getAttribute("title"));
</script>
</body>
</html>
```

5.5.2 Membuat elemen node melalui script

Method yang berperan untuk tujuan kita disini diantaranya:

- `createElement()`
- `cloneNode()`

Okay kedua method diatas mungkin tidak seperti membandingkan apple to apple. Tetapi pada akhirnya kita mampu menghasilkan node baru dengan method-method diatas.

```
<!DOCTYPE html>
<html lang="en">
<body>
  <div id="hasil"></div>
<script>
  var parent = document.querySelector("div[id=\"hasil\"]");
  var link = document.createElement("a");
  link.setAttribute("href", "http://kask.us/");
  link.textContent = "Cekibrot!";
  parent.appendChild(link);
```

```
</script>
</body>
</html>
```

Variable `parent` diatas ditetapkan sebagai wrapper node yang akan menampung elemen yang akan kita buat. Elemen yang akan kita buat adalah sebuah link atau dalam interface HTML tag **A**, kita berikan sebagai parameter berupa String untuk method `createElement()`. Karena link ini akan ditujukan ke suatu alamat URL, penambahkan attribute *href* pada elemen menggunakan method `setAttribute()`. Property `textContent` pada elemen bertujuan menampilkan text mewakili child bertipe `Text` bagi elemen link. Langkah terakhir meletakkan elemen ini menggunakan method `appendChild()` relative dibawah `parent` node.

Property `textContent` diatas merupakan salah satu cara membuat child bagi elemen link yang sudah dibuat. Bagaimana dan apa yang membedakannya dengan property lain seperti `innerHTML`?

Perbedaan yang cukup berarti bisa dilihat dengan menggantikannya dengan sebaris kode berikut:

```
link.innerHTML = "<b>Cekibrot!</b>";
```

Dengan property `innerHTML`, text dalam format markup HTML yang diberikan memungkinkan untuk diterjemahkan dan terbentuk menjadi suatu DOM. Dari potongan kode diatas, child node dari elemen link akan berupa elemen node **B** yang memuat suatu child bertipe `Text`.

Penggunaan property ini dinilai cukup praktis membangkitkan DOM melalui script, namun perlu diperhatikan dan sangat ekstra hati-hati terlebih jika nilai String yang dilewatkan pada property ini bersifat Dinamis.

Potensi serangan XSS (*cross-site-scripting*) pada halaman web bisa saja berawal dari sini. Sekedar saran 2-sen, sebisa mungkin jika tidak sangat terdesak, hindari penggunaan `innerHTML`, kecuali tahu betul apa yang sedang dilakukan. :)

5.6 Event