Översikt

- Document Object Model.

.easec

Document Object Model

- DOM.
- API.
- Att hitta element.
- Att hitta element snabbare.

.easec

DOM

- Alla moderna webbläsare implementerar en objektmodell kallad för DOM, framtaget av W3C.
- Representerar struktur för en webbsida.
- Tillhandahåller ett API som gör det möjligt att skriva JavaScript kod för att hantera sidan.
- Defininerar egenskaper som JavaScript kan förändra för ett element och vad som kan göras med sidan.
- DOM är designat för att vara oberoende av programmeringsspråk, men webbläsare har stöd för JavaScript.

.eavec

API

- Moderna webbläsare ger tillgång till följande API:er:

API	Funktion
DOM Core	Definierar basgränssnitt för att få tillgång till dokument, element, attribut eller text på sidan.
DOM Event Model	Basuppsättning för UI och händelser knutna till sida, innehåller även händelser för sidan som hanterar hur du kan lägga och ta bort händelser.
DOM Style Model	Definierar hur CSS-inställningar kan nås och hanteras.
DOM Traversal and Range Model	Travarserar element på sida för att manipulera dessa gemensamt.
DOM HTML API	Definierar objekt och metoder som representerar element på sidan.
DOM Load and Save	Definierar serialisering och deserialisering till XML.
DOM Validation API	Innehåller metoder för att dynamiskt uppdatera sida så de blir validierade mot HTML 4.01 specifikationen.

.eavec

Att hitta element

- Efter det att sidan har blivit inladdad, är den vanligaste metoden att hitta ett eller flera element, via DOM.
- DOM representerar de olika delarna i ett dokument som ett antal arrays:
  - *Forms array*, innehåller detaljer om formulär.
  - *Images array*, innehåller detaljer om bilder.
  - *Links array*, innehåller detaljer om länkar.
  - *Anchors array*, innehåller detaljer om alla `<a>` tags.
  - *Applets array*, innehåller detaljer om alla applets.

.eavec

Att hitta element (forts.)

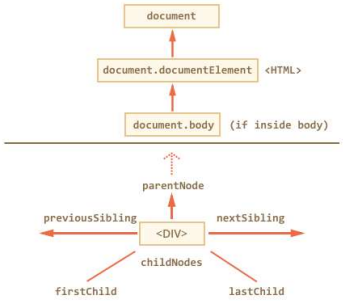
```
<form name="kontaktForm">
        <input type="text" name="namnBox" id="namnBoxId" />
</form>

// Tillgång till DOM-objekt som representerar formuläret
document.forms[0]
document.forms["kontaktForm"]
document.forms.kontaktForm
document.kontaktForm

// Tillgång till DOM-objekt som representerar namnBox
document.forms.kontaktForm.elements[0]
document.forms.kontaktForm.elements["namnBox"]
document.forms.kontaktForm.namnBox
document.kontaktForm.namnBox
```

.easec

Trädstruktur



.easec

Att hitta element

- Efter det att sidan har blivit inladdad, är den vanligaste metoden att hitta ett eller flera element, via DOM.
- DOM representerar de olika delarna i ett dokument som ett antal samlingar (eng: Collections):
  - *Elements*, innehåller detaljer om element.
  - *HTML attribute*, innehåller detaljer attribut.
  - *Text inuti HTML*, innehåller detaljer om text.
  - *Kommentarer*, innehåller detaljer om kommentarer.

.easec

Struktur

- De översta tre noderna är nårbara direkt som egenskaper för `document`.

  `<html>` = `document.documentElement`

  `<body>` = `document.body`

  `<head>` = `document.head`

- Child nodes är element som är en direkt underordnad nod, exempelvis `<head>` och `<body>` är underordnad `<html>`.
- Descendants är alla fallande under ett givet element.

.eavec

Struktur -exempel

```
<!DOCTYPE html>
<html lang="sv">
          <head>
             <meta charset="utf-8" />
             <title>Struktur</title>
          </head>

<body>
  <div>Starta</div>

  <ul>
    <li>
      <b>Information</b>
    </li>
  </ul>
</body>
</html>                                                    modul4demo1.html
```

- Frågar vi efter alla descendants för `<body>`, kommer vi att få `<div>`, `<ul>` och alla nestade element såsom `<li>` och `<b>`.

.eavec

Att tänka på

- `document.body` **kan vara** `null`, dvs finns inte.
- Skript kan inte få tillgång till något som inte finns när det körs.

```
<!DOCTYPE html>
<html lang="sv">
          <head>
             <meta charset="utf-8" />
             <title>modul4demo2</title>
             <script>
                alert( "Från HEAD: " + document.body ); // null, finns ingen <body> i nu.
             </script>
          </head>

<body>

  <script>
    alert( "Från BODY: " + document.body ); // HTMLBodyElement, existerar nu.
  </script>

</body>
</html>                                                    modul4demo2.html
```

.eavec

firstChild och lastChild

- `firstChild` och `lastChild` är genväg till första respektive sista underordnade elementet.
- Funktionen `hasChildNodes()` kan användas för att kontrollera om underordnade element finns.

```
elem.childNodes[0] === elem.firstChild
elem.childNodes[elem.childNodes.length - 1] === elem.lastChild
```

.eo/ec

DOM collection

- Vid första blicken ser `childNodes` ut som array, men är en speciell typ av collection.
- Två saker att tänka på:
  - Vi kan använda `for..of`.

```
for (let node of document.body.childNodes) {
  alert(node); // visar alla noder från collection
}
```

  - Vi kan inte använda metod för array.

```
alert(document.body.childNodes.filter); // undefined (det finns
ingen filter metod!)

alert( Array.from(document.body.childNodes).filter ); // kan
användas
```

.eo/ec

DOM collection (forts.)

- DOM collection är **endast läsbara**, gäller även egenskaper för navigation.
- Andra metoder måste användas för att förändra.
- DOM collection är **levande**.
- **Använd inte** for..in loop, då DOM collection innehåller egenskaper som inte används.

.eo/ec

Att tänka på -exempel

```
<!DOCTYPE html>
<html lang="sv">
          <head>
             <meta charset="utf-8" />
             <title>modul4demo3</title>
          </head>

<body>
  <div>Starta</div>

  <ul>
    <li>
      <b>Information</b>
    </li>
  </ul>

<script>
  // visa 0, 1, length, item, values etc.
  for(let prop in document.body.childNodes) alert(prop);
</script>
</body>

</html>                                                    modul4demo3.html
```


Att tänka på -exempel

```
<!DOCTYPE html>
<html lang="sv">
          <head>
             <meta charset="utf-8" />
             <title>modul4demo4</title>
          </head>

<body>
  <div>Starta</div>

  <ul>
    <li>
      <b>Information</b>
    </li>
  </ul>

<script>
  // visa 0, 1, length, item, values etc.
  for(let prop of document.body.childNodes) alert(prop);
</script>
</body>

</html>                                                    modul4demo4.html
```


Siblings

- Siblings eller syskon är noder som har samma föräldrer.
- Exempelvis:

  `<head>` och `<body>` är syskon, utgår från samma föräldrer.
- Föräldrer är tillgänglig som `parentNode`.
- Nästa nod med samma föräldrer är tillgänglig med `nextSibling` och föregående nod är tillgänglig som `previousSibling`.

Siblings -exempel

```
<!DOCTYPE html>
<html lang="sv">
<!DOCTYPE html>
<html><head></head><body><script>
  // HTML har tendens att lägga till extra tomma text noder.

  // föräldrer av <body> är <html>
  alert( document.body.parentNode === document.documentElement ); // true

  // efter <head> kommer <body>
  alert( document.head.nextSibling ); // HTMLBodyElement

  // innan <body> kommer <head>
  alert( document.body.previousSibling ); // HTMLHeadElement
</script></body></html>                                    modul4demo5.html
```

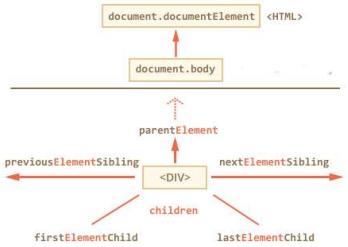
.eaec

Att arbeta endast med element

- De egenskaper som vi har tittat på hittills visar alla noder, textnoder, elementnoder och kommentarsnoder.
- Men för många arbetsuppgifter är vi bara intresserade av att manipulera elementnoder, som representerar tags och struktur för dokumentet.

.eaec

Att arbeta endast med element (forts.)



.eaec

Att arbeta endast med element -exempel

```
<html lang="sv">
        <head>
                <meta charset="utf-8" />
                <title>modul4demo6</title>
        </head
  <body>
  <div>Starta</div>

  <ul>
    <li>Information</li>
  </ul>

  <div>Slut</div>

  <script>
    for (let elem of document.body.children) {
      alert(elem); // DIV, UL, DIV, SCRIPT
    }
  </script>
  ...
</body>
</html>
```

modul4demo6.html

.eavec

Att arbeta med element med många egenskaper

- `<table>` är ett exempel på element med många egenskaper:
  - `table.rows`
  - `table.caption/tHead/tFoot`
  - `table.tBodies`
- `<thead>`, `<tfoot>` och `<tbody>` tillhandahåller egenskapen rows.
- `tbody.rows` är en samling av `<tr>`.

.eavec

Att arbeta med element med många egenskaper (forts.)

- `<tr>`: innehåller:
  - `tr.cells`, samling av `<td>` och `<th>` celler inuti given `<tr>`.
  - `tr.sectionRowIndex`.
  - `tr.Row.Index`.
- <td> och <th>, innehåller:
  - td.cellIndex.

.eavec

Att hitta element snabbare

- Som ett alternativ definierar DOM metoder på objektet `document` som inhämtar element baserat på attributen `ID` eller `Name`.
- `document.getElementById(IdString)`, returnerar ett element identifierat av id.
- `document.getElementByName(NameString)`, returnerar ett element identifierat av namn.

```
var anvNamnBox = document.getElementById(”namnBoxId”);
var anvNamn =  namnBox.value;
```

.eosec

Att hitta element snabbare -exempel

```
<!DOCTYPE html>
<html lang="sv">
    <head>
        <meta charset="utf-8" />
        <title>modul3demo7</title>
    </head>
<body>

<form id="minBilForm">
  Favorit bilmärke: <input type="text" name="fname" value="Volvo">
</form>

<form id="minFavoritFärgForm">
  Favorit färg: <input type="text" name="favfarg" value="Blå">
</form>

<p>Klicka på knappen för att visa namn för varje form element i document.</p>

<button type="button" onclick="minFunktion()">Klicka mig</button>

<p id="demo"></p>

<script>
function minFunktion() {
    var x = document.forms;
    var txt = "";
    var i;
    for (i = 0; i < x.length; i++) {
        txt = txt +  x[i].id + "<br>";
    }
    document.getElementById("demo").innerHTML = "Formulär:" + "<br>" + txt + "<br>" + "Antal formulär:" + "<br>" + x.length;
}
</script>

</body>
</html>
```

Favorit bilmärke: Volvo

Favorit färg: Blå

Klicka på knappen för att visa namn för varje form element i document.

Klicka mig

minBilForm
minFavoritFärgForm

modul4demo7.html

.eosec

Övning Arbeta med DOM

.eosec

Repetitionsfrågor