



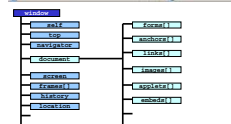
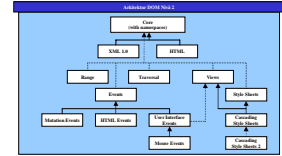
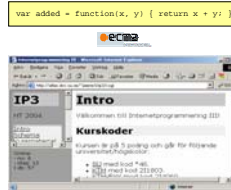
JavaScript och DOM

Introduktion

1

Introduktion

Disposition



2

Introduktion

Core JavaScript

- Språket

- Inbäddat
- Interpreterande
- Generellt
- Stort och kraftfullt
- Likheter med språk som C, C++, Java, (LISP?)
- JavaScript är inte Java ☹

```
function add(x, y) {  
  return x + y;  
}
```



3



Introduktion

Core JavaScript

- Standard

- Definieras av *European Computer Manufacturers Association* (ECMA)

- Implementation

- JavaScript i webbservrar från Netscape
- JScript i webbservrar från Microsoft

4



Introduktion

Core JavaScript

- EcmaScript 1

- Första standarden av språket JavaScript
- Standardisering av grundläggande koncept i JavaScript 1.1
- Dock ej av switch-satsen, inget stöd för reguljära uttryck
- Några extra finesser

- EcmaScript 2

- Förtydligande av tidigare, inga nya funktioner

- EcmaScript 3

- Switch-sats, reguljära uttryck, undantagshantering

5

Introduktion

Client-Side JavaScript

- Språket i ett sammanhang

- Implementerat i webbläsare
- Exekveringsmiljö på klientsidan
- Exekverbart innehåll
- Funktion och dynamik
- Statiska webbsidor blir interaktiva program

6

Introduktion

Client-Side JavaScript

- JavaScript 1.0 (JScript 1.0)
 - Webbbläsare: NN 2, IE 3
 - Språk: buggigt
 - DOM: Mycket begränsad (formulär)
- JavaScript 1.1 (JScript 2.0)
 - Webbbläsare: NN 3, IE 3
 - Språk: Array-objekt
 - DOM: Något bättre (bildhantering)

7

Introduktion

Client-Side JavaScript

- JavaScript 1.2
 - Webbbläsare: NN 4
 - Språk: Nästan EcmaScript 1
 - switch-satsen, regulära uttryck m.m.
 - DOM: Dålig, men visst stöd för DHTML (JavaScript + CSS) och hantering av olika lager

8

Introduktion

Client-Side JavaScript

- JavaScript 1.3 (JScript 3.0)
 - Webbbläsare: NN 4.5, IE 4
 - Språk: EcmaScript 1
 - DOM
 - NN: Samma dåliga
 - IE: Mycket bra, allt skriptbart!

9

Introduktion

Client-Side JavaScript

- JavaScript 1.4 (JScript 4.0 finns ej)
 - Serversidan, Netscape server-produkter
- JScript 5.0
 - Webbbläsare: IE5
 - Språk: Nästan EcmaScript 3
 - Undantagshantering
 - DOM: DHTML med `document.all[]`

10

Introduktion

Client-Side JavaScript

- JavaScript 1.5 (JScript 5.5-5.6)
 - Webbbläsare: Mozilla, NN 6, IE 5.5-6.0
 - Språk: EcmaScript 3
 - undantagshantering
 - DOM
 - NN/Mozilla: Stödjer W3Cs standard i stort
 - IE: Partiellt stöd för W3Cs standard

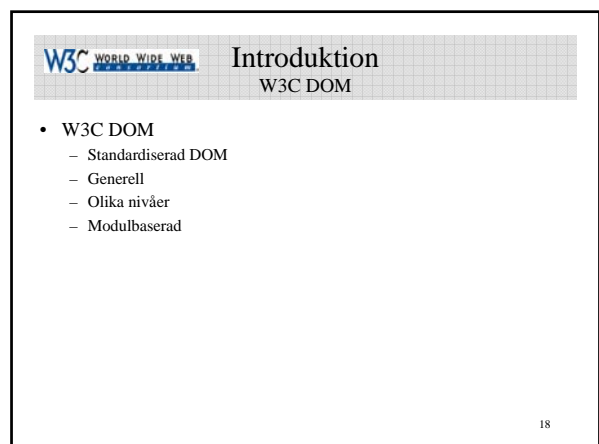
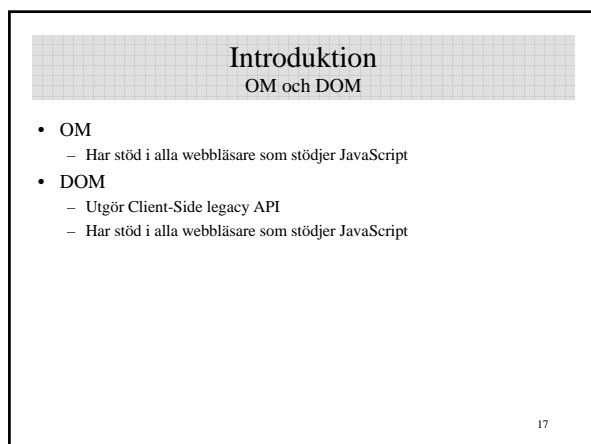
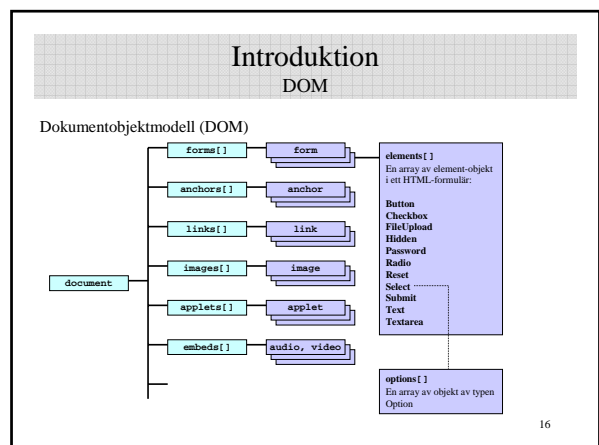
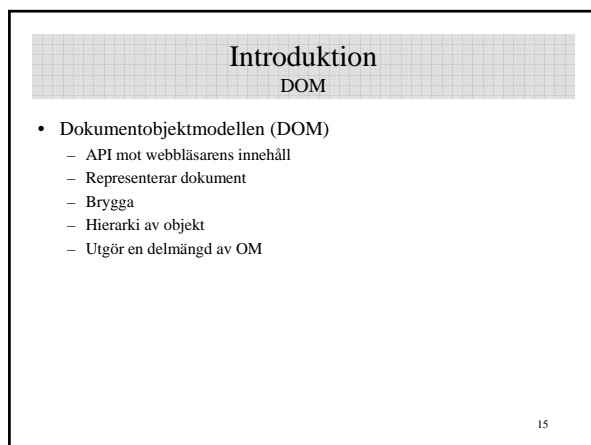
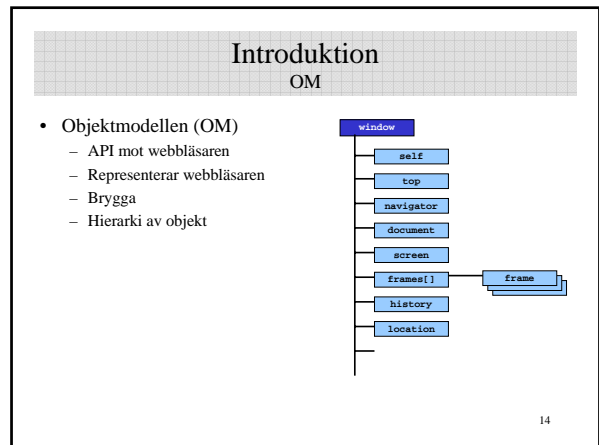
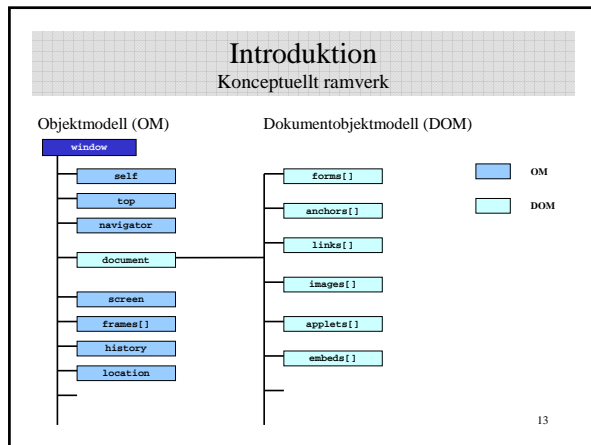
11

Introduktion

Client-Side JavaScript

- Implementation
 - JavaScript i webbläsare från Netscape
 - JavaScript i vissa övriga webbläsare
 - JScript i webbläsare från Microsoft
- En tolk för språket JavaScript
- En motor för att exekvera JavaScript
- Ett konceptuellt ramverk
 - Objektmodell (OM)
 - Dokumentobjektmodell (DOM)

12

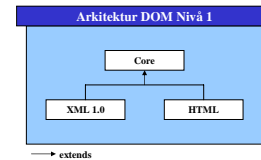


Introduktion W3C DOM

- Standarden definieras av *World Wide Web Consortium (W3C)*
 - DOM Nivå 0
 - Begränsad dokumenthantering och händelsehantering
 - Client-Side legacy API (gamla DOM)
 - Samlingsnamn för alla tidigare vilda försök
 - Ingår som formaliserad del av DOM Nivå 1
 - Ska garantera fortsatt framtida stöd
 - DOM Nivå 1 [1998-10-01]
 - Utbyggd dokumenthantering
 - DOM Nivå 2 [2000-11-13]
 - Avancerad händelsehantering, CSS, iteratorer m.m.
 - DOM Nivå 3 [klar ?]
 - Tangent-händelser m.m.

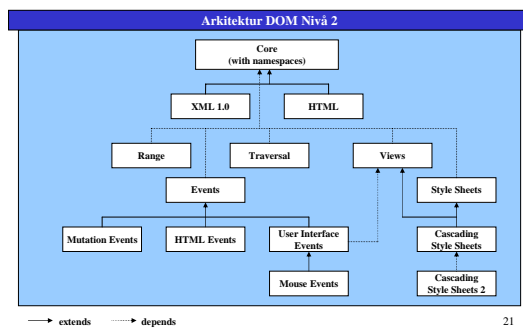
19

Introduktion W3C DOM



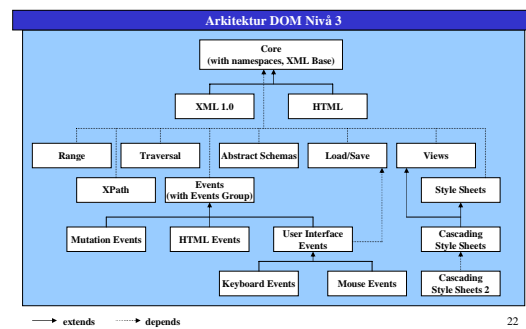
20

Introduktion W3C DOM



21

Introduktion W3C DOM



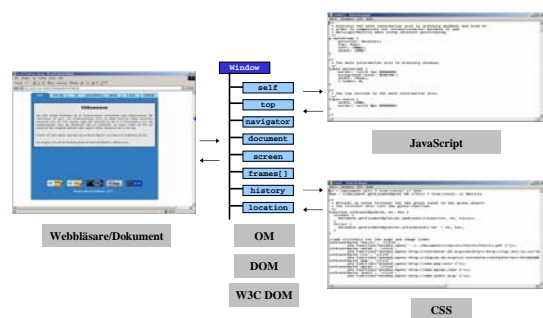
22

Introduktion W3C DOM

- Hur är stödet för W3C DOM i webbläsare?
 - Mozilla
 - Bäst, närmast DOM 2,
 - Identisk rendering/funktion i Linux, Win32, MacOS
 - NN
 - Ok, de flesta av modulerna i DOM2
 - IE4+
 - Bra, stödjer det mesta av DOM 1 och delar av DOM 2
 - Core, HTML, CSS tillräckligt
 - Stödjer inte DOM2 händelsemodul
 - Olika rendering/funktion i Linux, Win32, Mac
 - Även Opera, KDE/Conquer mfl ok

23

Introduktion Översikt



24

Introduktion

Ett exempel

En JavaScript-fil

```
/*
 * For each count up to the given total, a new node is created with a
 * text element containing the result. The element is added as a new
 * child to the given parent node.
 */
function count(total, parent) {
  var msg = "Nummer: ";
  var p, data;
  for(var i = 1; i < total + 1; i++) {
    p = document.createElement("p");
    data = document.createTextNode(msg + i);
    p.appendChild(data);
    parent.appendChild(p);
  }
}
count(10, document.getElementById("count"));
```

Funktion

Anrop

Introduktion

Ett exempel

Ett dokument med HTML

```
<html>
<head>
  <title>
    En räknare
  </title>
</head>
<body>
  <h2>En uppräknings</h2>
  <div id="count"></div>
  <script type="text/javascript" src="exempel1.js"></script>
</body>
</html>
```

Exempel 1

MIME

URL

26

Introduktion

Kompabilitet

- Plattformar och webbläsare
- Webbläsarnas implementering av
 - Motor för JavaScript
 - OM och DOM
- Webbläsarnas olika versioner

27

Introduktion

Inläring



- Övergripande kunskaper
- Förstå strukturer
- Känna till olika objekt
- Var öppen och våga
- Man lär sig inte detta innantill ☺
- Små steg
- Lek och öva, gör det roligt!
- Ha tålamod
- Utnyttja referensguider!
- Utnyttja länkar på kursens hemsida

28

Introduktion

Inläring

- Core JavaScript
 - Språkets grunder
 - Core JavaScript Reference (API)
- Client-Side JavaScript
 - Händelsestyrd programmering
 - OM
 - DOM (mindre viktig)
 - Client-Side JavaScript Reference (API)
 - W3C DOM
 - W3C DOM Reference

Avsnitt 2-11
Sid. 427-537

Avsnitt 12.1,3 och 19
Avsnitt 12-13
Avsnitt 14-16
Sid. 541-681
Avsnitt 17-19
Sid. 685-853

29

Core JavaScript



30

Core JavaScript

- Språket JavaScript
 - Interpreterande
 - Skiftlägeskänsligt
 - Otypat
 - Typomvandling
 - Funktioner
 - Objektbaserat
 - Implementerar olika objekt
 - Objektorientering?



31

Core JavaScript

Lexikal struktur

- UNICODE (16 bitar)
- Skiftlägeskänsligt
 - Nyckelord, variabler, namn på funktioner m.m.
 - while måste skrivas "while", inte "While" eller "WHILE"
- Whitespaces och radbrytningar
 - Ignoreras mellan tokens
 - En token är ett nyckelord, namn på variabel eller funktion, nummer etc.
 - Uppmuntrar till strukturerad kod!
- Semikolon
 - Separation av satser
 - Kan uteslutas om satserna står på skilda rader (ej god sed)

```
a = 3
b = 3
```

```
a = 3; b = 3
```



(Avsnitt 2)

32

Core JavaScript

Lexikal struktur

- Kommentarer

```
// En rad med kommentar
```

```
/* En annan rad */
```

```
/*
 * Ytterligare en kommentar
 * fast med flera rader
 */
```

33

Core JavaScript

Lexikal struktur

- Literaler - datavärden

```
12
true
```

```
"hello world"
null
```

- Identifierare – namnger variabler, funktioner
 - Första tecknet måste vara en bokstav, _, eller \$
 - Efterföljande kan vara bokstäver, siffror, _, eller \$

```
i
my_doom
```

```
_element_16
$myValue
```

- Reserverade ord
 - Kan inte användas för att namnge variabler, funktioner etc.

```
break if for null while
```

34

Core JavaScript

Variabler

- Otypade

```
a = 10;      a = new Array(5);
b = "ten";   b = a;
```

(Avsnitt 4 och 11.2-11.4)



- Kan lagra vad som helst
 - Primitiva typer
 - Datasamlingar
 - Funktioner
 - Andra objekt

35

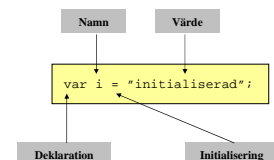
Core JavaScript

Variabler

- Deklaration och initialisering
 - Deklaration bör ske före användning
- Initialisering

```
var i;      var i, sum;
```

```
i = "initialiserad";
```



36

Core JavaScript Variabler

- Några allmänna saker
 - Permanenta variabler
 - Upprepad deklaration av samma variabel
 - Odeklarerade variabler:
 - Existerar inte
 - Läsa medför fel
 - Tilldela medför implicit deklaration (globalt)
 - Man bör alltid deklarera!
- null
 - Anti-värde
 - Innehåller inte ett objekt, en array, ett heltal etc.
- undefined
 - Deklarerad men ej initialiserad
 - En egenskap som inte finns

```
alert(u);  
u = 3;
```

```
var x;
```

37



Core JavaScript Variabler

- Räckvidd
 - Region och deklaration
 - Globala variabler (utanför funktioner) `var i = 10;`
 - Lokala variabler (inom funktioner) `var i = 10;`
 - Ingen blockräckvidd!

```
function contest(a) {  
  var i = a; //a lokal, precis som i, j och k  
  //i definierad genom hela funktionen  
  if (i == 0) {  
    var j = 1; //j definierad i resten av funktionen  
  } else {  
    var k = 2; //k definierad i resten av funktionen  
    document.write(k);  
  }  
  document.write(k); //k är fortfarande tillgänglig  
  document.write(j); //j tillgänglig, men ej initialiserad?  
}
```

38

Core JavaScript Variabler

```
var scope = "global"; //top-level code: global variabel  
function checkScope() {  
  var scope = "local"; //deklarera lokal variabel  
  document.write(scope); //den lokala variabeln används  
}  
checkScope(); //skriver ut "local"  
  
scope = "global"; //deklarera globalt utan var  
function checkScope() {  
  scope = "local"; //oops  
  document.write(scope); //den globala variabeln används  
  newScope = "local?"; //implicit deklaration av global variabel  
  document.write(newScope); //använder den nya globala variabeln  
}  
checkScope(); //skriver ut "locallocal"  
document.write(scope); //skriver ut "local"  
document.write(newScope); //skriver ut "local"
```

39

Core JavaScript Variabler

- Operationer på värden
 - Kopiering
 - Argument till funktioner
 - Jämförelser
- Hur sker dessa?
 - Genom värdesemantik
 - Värdet som räknas
 - Värdet kopieras till en ny variabel
 - Jämförelse sker genom värde
 - Genom referens
 - Endast en kopia
 - Referensen tilldelas och manipuleras
 - Jämförelse sker mot minnesutrymmet

40

Core JavaScript Variabler

- Värdesemantik

```
var x = 3, y = 4;
```

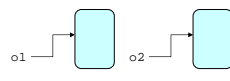
x 3 y 4

```
x = y;
```

x 4 y 4

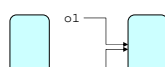
- Referens

```
var o1 = new Object();  
var o2 = new Object();
```



```
o1 = o2;
```

Oåtkomligt
utrymme



41

Core JavaScript Variabler

- I JavaScript
 - Primitiva datatyper genom värdesemantik
 - Referenstyper genom referens
 - Gäller även vid argument till funktioner
- Primitiva datatyper
 - Numeriska
 - Strängar
 - Boolean
- Referenstyper
 - Objekt
 - Arrayer och funktioner

42

Core JavaScript

Variabler

- En fil i JavaScript

```
var collection = new Array(10); //global

function saveDate(index, date) {
  index++; //lokal kopia ändras 2
  date.setMonth(10); //objektet ändras
  collection[index] = date;
}

function createDate() {
  var i = 1; //lokal
  var d = new Date(); //lokal
  saveDate(i, d); //värdesemantik, referens
}

createDate();
```

43

Core JavaScript

Operatorer

- Ungefär som i Java

- Styrka - precedens (*precedence*)



(Avsnitt 5)

- Aritmetiska	+ - * / % ++ --
- Likvärdighet	== ===
- Jämförelser	< > <= >=
- Logiska	&& !
- Tilldelning	= += -=
- Villkor	(x < 0) ? x = 1 : x = 10
- Arrayåtkomst	[]
- Funktionsanrop	()
- in	
- instanceof	
- typeof, new, delete	

44

Core JavaScript

Satser och kontrollstrukturer

- Enkla satser

```
var name = "Donald";
```



(Avsnitt 6)

- Sammanstatta satser { }

```
{
  name = "Donald";
  age = "35";
}
```

45

Core JavaScript

Satser och kontrollstrukturer

- Andra satser

```
return;
break;
continue;
with (object) statement
```

46

Core JavaScript

Satser och kontrollstrukturer

- Selektion if

```
if (a == 1) {
  //utför någonting
} else if (a == 2) {
  //utför något annat
} else {
  //för alla andra fall
}
```

47

Core JavaScript

Satser och kontrollstrukturer

- Selektion switch

```
switch (n) {
  case 0:
    //utför någonting
    break;
  case 1:
    //utför någonting
    break;
  default:
    //om inget annat matchar
    break;
}
```

48

Core JavaScript

Satser och kontrollstrukturer

- Iteration `for`

```
for (int i = 0; i < 10; i++) {  
  document.write(collection[i]);  
}
```

```
for (int i = 0; i < collection.length; i++) {  
  var column = collection[i];  
  for (int j = 0; j < column.length; j++) {  
    document.write(collection[i][j]);  
  }  
}
```

49

Core JavaScript

Satser och kontrollstrukturer

- Iteration `while`

```
while (true) {  
  document.write("Oändlig loop");  
}
```

```
while (b) {  
  if (a == 1) {  
    continue;  
  }  
  if (a == 2) {  
    break;  
  }  
}
```

50

Core JavaScript

Primitiva datatyper

- Numeriska datatypen
 - Heltal, flyttal, hexadecimala tal, oktala tal
 - Alla värden representeras som flyttal



(Avsnitt 3.1-3.3)

Numeriska värden
0
170

3.14
.356787

6.02e23
5.678E-32

Användning
var i = 8;
var j = 7;
var x, y, z;

x = i + j;
y = i - j;
z = i*j;

51

Core JavaScript

Primitiva datatyper

- Textsträngar
 - En sekvens av tecken
 - Omsluten inom enkla eller dubbla citationstecken
 - Måste skrivas på en rad
 - Notera hur citationstecken används och tolkas

Strängar
"" //tom sträng
"en sträng"
'fungerar'
'en sträng"i en strängl"
"detta'också"
"We like O'Reilly's book"
"Strängen har\ntvå rader"
'En apostrof: O\'Reilly'

Användning
var msg;
var name = Donald;

//producerar "Hello world"
msg = "Hello " + "world";

msg = "Hej," + " " + name;

52

Core JavaScript

Primitiva datatyper

- Booleska datatypen
 - Representerar ett sanningsvärde
 - Endast två värden
 - Används ofta för styrning och jämförelser

Booleska värden
true
false

Användning
if (a == 4) {
 b = b + 1;
}

Uttrycket (a == 4) utvärderas.
Har a värdet 4 så utförs beräkningen.

Man kan tänka sig booleska värden som sanna eller falska,
på eller av, ja eller nej, 1 eller 0.

53

Core JavaScript

Funktioner

- En enhet av exekverbar kod
 - Inget åtkomstdeklaration (*public*, *private* etc.)
 - Returtyp deklaras ej (*void*, *String* etc.)
 - Kan ta emot argument (parametrar)
 - Kan returnera ett resultat
- Fördefinierade i implementering av JavaScript
- Egendefinierade



(Avsnitt 3.4 och 7)

54

Core JavaScript

Funktioner

- Användning av fördefinierade funktioner i JavaScript

```
var x = 25;  
var y;  
  
y = Math.sin(x);
```

```
var date = new Date();  
var time = date.getTime();
```

55

Core JavaScript

Funktioner

- Funktioner som syntaktiska satser
 - mest grundläggande definitionen
 - namn
 - argument
 - kropp inom { }
 - Returnera något?

```
function add(x, y) {  
    return x + y;  
}
```

```
var sum = 0;  
  
function add(x) {  
    sum += x;  
}
```

56

Core JavaScript

Funktioner

- Mer än bara syntax i JavaScript
 - Specialiserad form av objekt
 - De kan därmed ha egenskaper och metoder
 - Utgör värden som kan manipuleras
 - Kan lagras i variabler
 - Kan lagras som egenskaper hos objekt
 - Kan lagras i datasamlingar
 - Kan överföras som argument till andra funktioner

57

Core JavaScript

Funktioner

- Funktionskonstruktorn `new Function()`
 - Observera stort "F"
 - Dynamisk kompilering
 - Kompileras vid varje anrop!
 - Anonym

```
var f = new Function("x", "y", "return x + y;");
```

58

Core JavaScript

Funktioner

- Funktioner som literaler `function() { ...; };`
 - ECMAScript 3
 - Uttryck istället för sats
 - Anonym

```
var f = function(x, y) { return x + y; };
```

- För rekursiva operationer kan vi dock namnge dem

```
var f = function fact(x, y) { if (x <= 1000)  
    return 1; else return fact(x + y); };
```

59

Core JavaScript

Funktioner

- Vår funktion
 - Skapar ett nytt funktionsobjekt
 - Funktionen heter `add`
 - Må en variabel med namnet `add`

```
function add(x, y) {  
    return x + y;  
}
```

- En annan funktion

```
function operate(operator, opl, op2) {  
    return operator(opl, op2);  
}
```

- Wow!?

```
var i = operate(add, 2, 3);
```

60

Core JavaScript Funktioner

- Om antalet argument inte överensstämmer

- Objektet arguments
- En typ av array
- Egenskap hos funktioner
- Lagrar argumenten som skickas med

```
function add(x, y) {
  return x + y;
}
```

- Ett exempel

- Alla värden lagras i arguments

```
var sum = add(1, 2, 3);
```

- Värdet 3 åtkomligt i denna

```
arguments[2];
```

- Kan kontrollera antal argument

```
if (arguments.length != 2) {
  //utför någonting
}
```

61



Core JavaScript Objekt

- Sammansatt datatyp

- Egenskaper

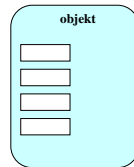
- Utgör variabler
- Kan lagra godtyckliga data
 - Datasamlingar
 - Funktioner
 - Andra objekt
- Kan läggas till dynamiskt

- Funktioner

- Kallas metoder
- Egenskapens namn utgör metodens namn



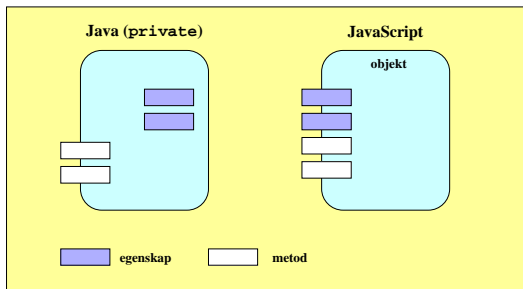
(Avsnitt 3.5-3.12 och 8)



62

Core JavaScript Objekt

- Åtkomst



Core JavaScript Objekt

- Konstruktor

- En funktion som anropas med new
- Ger referens till nytt objekt
- Initialiserar egenskaper

- Fördefinierade konstruktörer:

- Object()
- Date()
- RegExp()
- etc.

- Egna konstruktörer

- För en ny egen typ (klass) av objekt

64

Core JavaScript Objekt

- Skapa med new

```
var o = new Object();
var now = new Date();
var bool = new Boolean(true);
```

- Skapa som literal { name=value, ... , }

```
var me = {
  namn: "Donald Strömberg",
  ålder: 34,
  gift: false
};
```

65

Core JavaScript Objekt

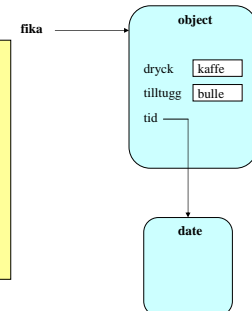
- Ett litet exempel

```
var fika = new Object();

//tilldela egenskaper
fika.dryck = "Kaffe";
fika.tilltugg = "Bulle";

//tilldela fler egenskaper
fika.tid = new Date();

//åtkomst
var d = fika.dryck;
var t = fika.tid.valueOf();
```



66

Core JavaScript Objekt

- Ett nytt exempel

```
function Fika(f, t) {  
  this.dryck = f;  
  this.tilltugg = t;  
  this.tid = new Date();  
}
```

Klass Fika

dryck
tilltugg
tid

```
var fika1 = new Fika("kaffe", "bulle");  
var fika2 = new Fika("thé", "mazarin");
```

67

Core JavaScript Objekt

- Metoder
 - Objektens funktioner
 - Anropas alltså hos objektet
 - Utgör egentligen egenskaper hos objekt

```
function bytDryck(d) {  
  this.dryck = d;  
}
```

```
var fika1 = new Fika("kaffe", "bulle");  
  
fika1.nyDryck = bytDryck;  
fika1.nyDryck("choklad");
```

68

Core JavaScript Objekt

- Mindre omständigt

```
function bytDryck(d) {  
  this.dryck = d;  
}  
  
function Fika(f, t) {  
  //egenskaper  
  this.dryck = f;  
  this.tilltugg = t;  
  this.date = new Date();  
  
  //metoder  
  this.nyDryck = bytDryck;  
}
```

69

Core JavaScript Objekt

- Globala objektet `global`
 - Skapas när JavaScript-tolken startar
 - Egenskaperna utgör globala variabler hos JavaScript-program
 - Initialiseras av tolken
 - Somliga är statiska medlemmar utan konstruktor: `Math`
 - Somliga utgör konstruktorfunktioner: `Date`, `Array`, `Function`, `RegExp`
 - Det är på så vis vi kan använda och/eller skapa dem ☺
- Egna globala variabler
 - Äterfinns utanför funktioner i JavaScript-filer
 - De tilldelas egentligen `global`
- Exekveringssammanhang
 - Globalt
 - Funktioner

70

Core JavaScript Objekt

Numerisk datatyp
`var x = 13;`
`var y = x.toString();`

Strängar
`var msg = "Hello";`
`var size = msg.toString();`
`var last_char = s.charAt(s.length - 1);`

Boolesk typ
`var bool = true;`
`var is_true = bool.valueOf();`

Hur fungerar detta egentligen?

71

Core JavaScript Objekt

- Omslutande klasser (*Wrapper objects*)
 - `Number`
 - `String`
 - `Boolean`
- Innehåller
 - Samma värde
 - Har egenskaper
 - Har metoder

- Skapas automatiskt

```
var s = "primitiv sträng";  
var x = s.length;  
var y = s.indexOf("prim")
```

- Kan skapas explicit

```
var x = new Number(10);  
var y = x.valueOf();
```

72



Core JavaScript

Datasamlingar

- Datasamling
 - En samling av data eller värden
 - En specialiserad form av objekt

(Avsnitt 3.6 och 9)



- Datasamlingar
 - Arrayer
 - Nästlade arrayer (matriser)
- Egenskaper
 - Värden kan vara av godtycklig typ
 - Positionerar värden i element
 - Elementet har nummer (*index*)
 - Indexeringen börjar från 0

73

Core JavaScript

Datasamlingar

- Array

```
var a = new  
Array();  
  
a[0] = 1.2;  
a[1] = "Hej";  
a[2] = true
```

0	1	2
1.2	"Hej"	true

```
var a = new Array(1.2, "Hej", true);
```

```
var b = a[0];  
a[3] = a[1];  
a[2] = 333;  
a[1] = { x:1, y:3 }; //object literal
```

74

Core JavaScript

Datasamlingar

- Array

```
var a = new Array(10);
```

0	1	2	3	4	5	6	7	8	9

```
var size = a.length;    size = ?
```

```
var width = document.images[1].width;
```

75

Core JavaScript

Datasamlingar

- ECMAScript 3

```
var a = [1.2, "JavaScript", true];
```

- Nästlade arrayer i ECMAScript3

```
var matrix = [[1,2,3], [4,5,6], [7,8,9]];
```

	0	1	2
0	1	4	7
1	2	5	8
2	3	6	9


```
x = matrix[1][2];    x = ?
```

76

Core JavaScript

Datasamlingar

- Godtyckliga uttryck i ECMAScript 3

```
var base = 10;  
var table = [base, base+1, base+3];
```

- Addera nya element
 - variabel storlek
 - ändra storlek dynamiskt
 - se upp med *sparse* (beroende av implementation)

```
var a = new Array();  
  
a[0] = 1;  
a[100] = "Element nr 100";
```

77

Core JavaScript

Objekt

- Användbara fördefinierade klasser:

- Object (generisk, alla "ärver" ifrån detta)
- Function (representerar funktioner)
- Number (wrapper)
- String (wrapper)
- Boolean (wrapper)
- Array (datasamling)
- Date (datum och tid)
- Math (matematiska operationer)
- RegExp (reguljära uttryck)
- Error (undantagshantering)

(Core JavaScript Reference sid 429-537)

Core JavaScript

Objektorientering

- Objektorientering hos klassbaserade språk

- Stark typning
- Klassmetoder och instansmetoder
- Klassvariabler och instansvariabler
- Reglerad synlighet och åtkomst *private, protected*
- Uttalad struktur för en klass *public class MyClass {}*
- Metoder och variabler definieras i strukturen, kan inte läggas till på annat sätt
- Ett koncept för en klasshierarki
- Klassbaserat arv *extends*

79

Core JavaScript

Objektorientering

- JavaScript följer en annan modell

- Svag typning
- Allt är öppet och åtkomligt
- Egenskaper och metoder kan läggas till dynamiskt
- Funktioner utgör objekt
- Därmed är även konstruktorn ett objekt
- Konstruktorn utgör en klassdefinition
- Prototypbaserat arv

- JavaScript simulerar koncepten hos klassbaserade språk

80

Core JavaScript

Objektorientering

- Prototyper

- Alla objekt har en prototyp
- Prototypen utgörs av en inbyggd egenskap
- Egenskapen heter *prototype*

- Detta gäller även

- Funktioner
- Därmed även konstruktorn

- Egenskaper och metoder kan tilldelas *prototype*
- Egenskaperna i en viss prototyp utgör även egenskaper i alla objekt med denna prototyp
- Därmed ärver objekt sina egenskaper och metoder från sin prototyp

81

Core JavaScript

Objektorientering

- Ännu mer abstrakt

- Prototypen definieras av konstruktorn som används för att skapa och initialisera objektet
- *prototype* är initialt tomt, förutom egenskapen *constructor* som refererar till den konstruktorn som initialiserar objektet
- Men det som definieras i det kommer att ärvas av alla objekt som skapas av konstruktorn

- För att uppnå arv

- Tilldela metoder, konstanter och andra egenskaper till konstruktorns *prototype*

82

Core JavaScript

Objektorientering

- Så hur simulerar JavaScript

- Klassmetoder och instansmetoder?
- Klassvariabler och instansvariabler?

- Klassmetoder

- Egenskap hos konstruktorn

- Instansmetoder

- Egenskap hos *prototype* i konstruktorn

- Klassvariabler

- Egenskap hos konstruktorn med hjälp av *this*

- Instansvariabler

- Egenskap hos *prototype* i konstruktorn

83

Core JavaScript

Objektorientering

```
function Circle(radius) { //konstruktorn definierar klassen
  //prototype = new Object();
  this.r = radius; //instansvariabel
}

Circle.PI = 3.14; //klassvariabel

function Circle_area() { //en vanlig funktion
  return Circle.PI * this.r * this.r;
}

Circle.prototype.area = Circle_area; //som vi gör till en instansmetod

function Circle_max(a, b) { /*
  if (a.r > b.r) return a; * en funktion som tar två cirklar
  else return b; * som argument
}

Circle.max = Circle_max; //som nu blir en klassmetod
```

84

Core JavaScript

Objektorientering

- Så hur fungerar arv i JavaScript?
 - Generiskt objekt `Object`
 - Objekt ärver egenskaper från objektet `prototype` i sin konstruktor
 - Men `prototype` är också ett objekt skapat med `new Object()` JavaScript skapar detta automatiskt varje gång en konstruktorfunktion anropas
 - Därför ärver `prototype` från `Object.prototype`
 - Alla nya objekt av klasser kan därmed ärva från `prototype`
 - Fördefinierade klasser i JavaScript är subklasser till `Object`

85

Core JavaScript

Objektorientering

- Hur uppnår man en mer komplex hierarki
 - Anta att vi skapar en egen subklass till `Object`

```
function Vehicle(n, c) {  
  this.name = n;  
  this.color = c;  
}  
//prototype = new Object();
```

- Sedan skapar vi en tänkt subklass

```
function Car(r) {  
  this.regNo = r;  
}  
//prototype = new Object();
```

- Slutligen överlagrar vi hur `prototype` ska skapas:

```
Car.prototype = new Vehicle();  
//prototype = new Vehicle();
```

86

Core JavaScript

Objektorientering

- Vi har dock tappat bort vår skapare
 - Vi skrev över prototypobjektet som JavaScript tillhandahåller
 - Därmed även egenskapen `constructor`
 - Denna refererar till den konstruktorfunktion som skapade objektet
 - Vi kan dock återställa ordningen

```
Car.prototype.constructor = Car;
```

87

Core JavaScript

Reguljära uttryck

- Beskriva textuella mönster
- Matcha mönster
- Sökning
- Ersättning
- Klassen `RegExp`
- ECMAScript och PERL



(Avsnitt 3.10 och 10)

88

Client-Side JavaScript



89

Client-Side JavaScript

- Funktion och dynamik
- Kontroll över
 - Webbbläsaren och tillstånd
 - Dokumentet och dess innehåll
- Interaktion med
 - Användaren
 - Formulär
 - Java Applets
- Dynamisk generering av innehåll
- Modifiera stilsättningsattribut i CSS
- Transformation (HTML/XML)

Exempel 2

Exempel 3

Exempel 4

Exempel 5



(Avsnitt 12-16)

90

Client-Side JavaScript

- Vad JavaScript inte kan göra
 - Generera grafiska gränssnitt
 - Läs från och skriva till filer
 - Nätverkskopplingar
- JavaScript bör inte
 - Utgöra enda kontrollen av inmatade värden
 - Ersätta säkerhetsmekanismer som bör ske på serversidan

91

Client-Side JavaScript Omgivning

"To understand client-side JavaScript you must understand the conceptual framework of the programming environment provided by the web browser."

JavaScript: The Definitive Guide, Forth Edition (O'Reilly, 2002)

- Händelsestyrd programmering
- Globala objektet
- Objektmodellen (OM)
- Dokumentobjektmodellen (DOM)
- DOM W3C

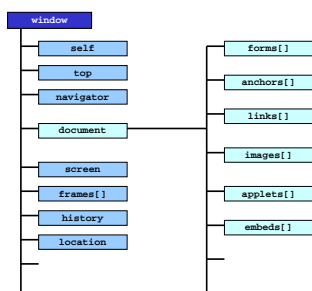
92

Client-Side JavaScript Omgivning

```
function add(x, y) {
  return x + y;
}
```

```
Object
Function
Number
Array
Date
Math
RegExp
{...}
```

Core JavaScript



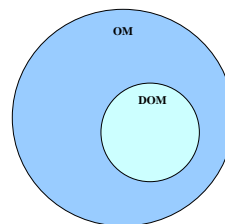
OM

DOM

93

Client-Side JavaScript Omgivning

- DOM är en delmängd av OM



94



Client-Side JavaScript W3C DOM

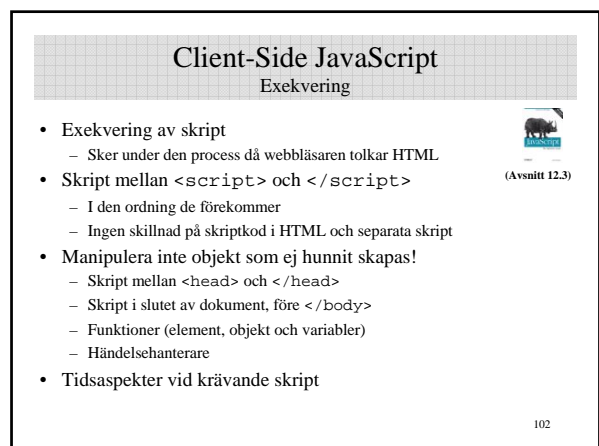
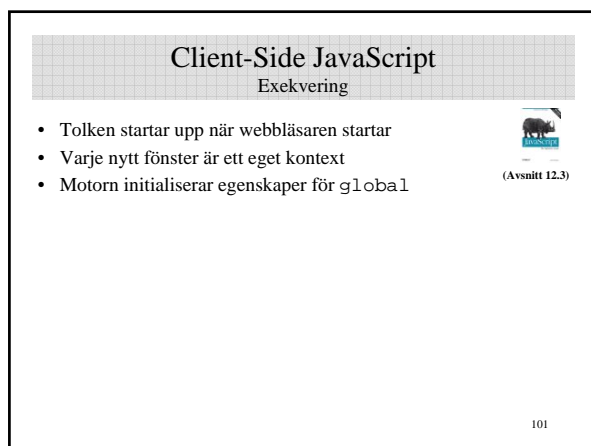
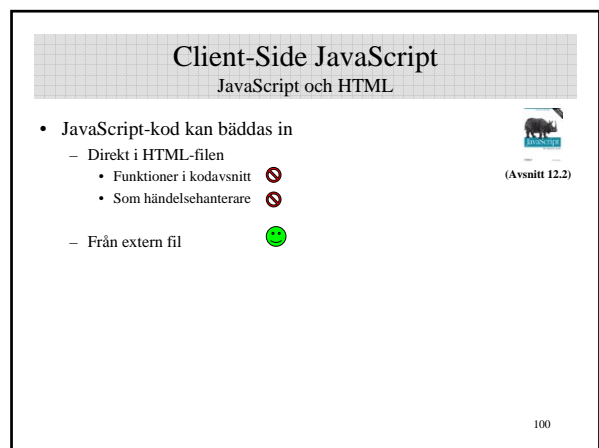
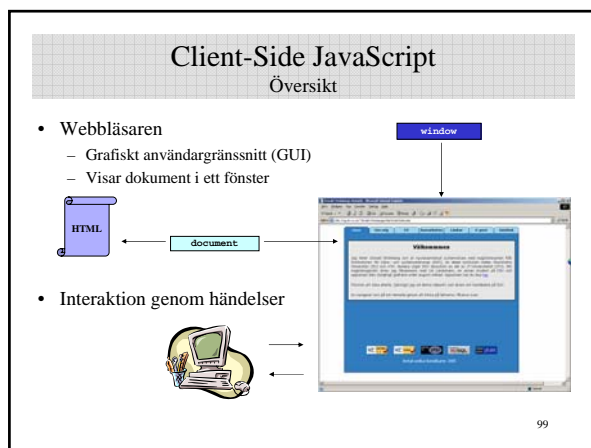
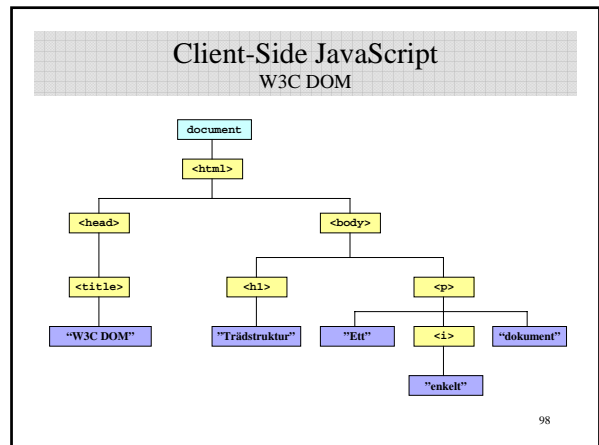
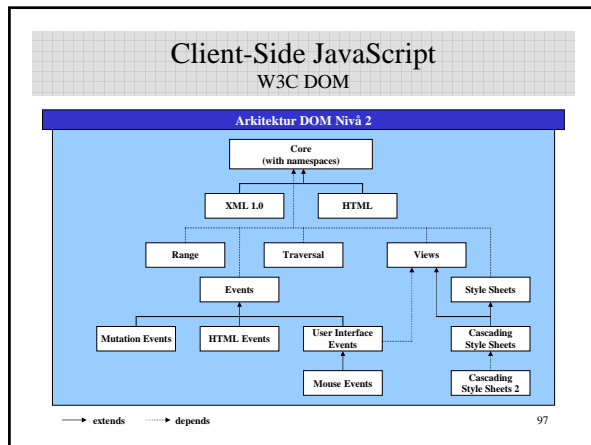
- Standardiserad DOM
- Generell
- Olika nivåer
- Modulbaserat
- Varje modul är ett API
- Dokument representeras som träd

95

Client-Side JavaScript W3C DOM

- DOM Nivå 0
 - Alla tidigare försök
- DOM Nivå 1
 - Utbyggd dokumenthantering
- DOM Nivå 2
 - Avancerad händelsehantering, CSS, iteratorer m.m.
- DOM Nivå 3
 - Tangent-händelser m.m.

96



Client-Side JavaScript

Exekvering

- Var lägger vi länkarna till skripten?

```
<html>
<head>
  <title>Händelsehantering</title>
</head>
<body>
  <script type="text/javascript" src="functionDefinitions.js" />
  <h2>Ett formulär</h2>
  <form action="" method="">
    <input type="button" name="alert" value="Klicka här" />
  </form>
  <script type="text/javascript" src="event.js" />
  <script type="text/javascript" src="functionOperations.js" />
</body>
</html>
```

103

Client-Side JavaScript

The Event-Driven Programming Model

- Förr
 - Batch
 - Textbaserade terminaler
 - Sammanhängande kodblock
 - Kontrollerade flöden
 - Körde från start till slut
 - JavaScript kan köras så
- Idag
 - Grafisk omgivning
 - Pekdon och tangentbord
 - Program reagerar på asynkron input

104

Client-Side JavaScript

The Event-Driven Programming Model

- Händelsestyrd programmeringsmodell
 - Händelsehanterare (*eventhandler*) registreras
 - Händelsehanterare definierar en respons
 - Input sker
 - En händelse genereras (*event*)
 - Applikationen informeras
 - Händelsen vidarebefordras till lämplig händelsehanterare
 - Responsen exekveras
 - Hanterare anropas alltså av applikationen vid behov
- Gäller alla grafiska användargränssnitt
- Ett GUI startar upp och väntar...

105

Client-Side JavaScript

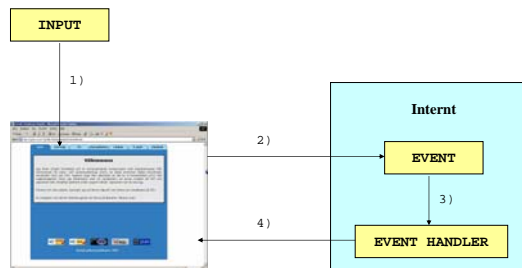
The Event-Driven Programming Model

- Webbbläsaren
 - Genererar en händelse när exempelvis
 - Ett dokument har laddats klart
 - Användaren stänger applikationen
 - När användaren klickar på en länk
 - När användaren klickar på en knapp för att sända ett formulär
 - Vissa av dessa tas om hand automatiskt
 - Men ibland vill vi styra det själva!



Client-Side JavaScript

The Event-Driven Programming Model



107

Client-Side JavaScript

The Event-Driven Programming Model

- Tre viktiga aspekter av händelsehantering
 - Definition och registrering av händelsehanterare
 - Propagering av händelser
 - Interaktion med händelser

108

Client-Side JavaScript

The Event-Driven Programming Model

- Händelsehantering
 - The original model (OM/DOM)
 - The standard event model (W3C DOM Nivå 2)
 - The Internet Explorer event model
 - The Netscape 4 event model
- Viktiga frågor
 - Hur fungerar händelsehantering i respektive?
 - Kompatibilitetsproblem? Kan de kringgås?
 - Kan vi följa standarden XHTML?
 - Kan vi använda DHTML och separera delarna?

109

OM



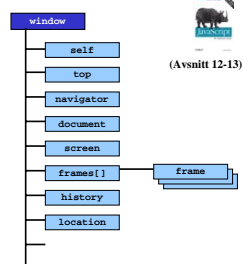
110

OM

Objekthierarkin

- Objektmodellen (OM)
 - API mot webbläsaren
 - Hierarki av objekt
 - Stöd i alla webbläsare

```
var frame = window.frames[0];
window.close();
```



111

OM

Objekthierarkin

- Ger åtkomst och information
 - Historik
 - Webbläsaren
 - Bildskärmen
- Möjliggör kontroll och interaktion
 - Mellan fönster/ramar
 - Öppna/stänga nya fönster
 - Dialogfönster
 - Omdirigera till andra sidor

112

OM

Händelsehantering

- Händelsehantering i OM
 - Timeout
 - Intervall
 - Viss undantagshantering med onerror
- Mycket begränsad
- Ren händelsehantering?

113

OM

Objekten

- Utvalda objekt kommer att tas upp



- Se upp med egenskaper och metoder specifika för olika webbläsare

114


OM Window

- Representerar webbläsarens fönster
- Fönsterhantering
- Globala objektet
 - Alla globala variabler är egenskaper till window
 - Även de som ni definierar!
 - Jämför med global i Core JavaScript

(Avsnitt 13.1-13.5, 13.8)

```
var global = 42;
window.global = 42;
```

Ekvivalent!



OM Window

Några viktiga egenskaper

- self
- top
- navigator
- document
- screen
- frames[]
- history
- location

Några andra egenskaper

```
closed //är fönstret stängt?
status //texten i statusraden
self //synonym för window
opener //vem sköt katten?
onerror //enkel händelsehantering
```

Några användbara metoder

```
open() //öppna nytt fönster
close() //stäng fönster
alert() //litet dialogfönster
resizeBy() //ändra storlek
```

116

OM Window

- Några exempel

```
window.alert("En varningsruta");
```

```
window.close();
```


Exempel 6

```
window.open('../window.html', 'new_window',
'height=260,width=630,resizable=yes,scrollbars=yes');
```

Exempel 7

```
window.open('../cv.pdf');
```

Exempel 8



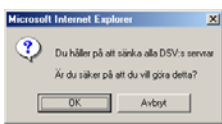
117

OM Window

- Ett exempel till

```
var msg = "\nDu håller på att sänka alla DSV:s  
servrar\n\nÄr du säker på att du vill göra  
detta?";

if (confirm(msg)) {
//om användaren klickar på "OK"
} else {
//om användaren klickar på "Avbryt"
}
```



118

OM Window

- Fördröjning och intervall
- Felhantering med egenskapen onerror

Några användbara metoder

```
setTimeout() //sätt tidpunkt för händelse
clearTimeout() //radera tidpunkt
setInterval() //som setTimeout() fast repetition
clearInterval() //ta bort repetition
```

```
setTimeout("window.alert('5 sekunder har passerat')", 5000);
```

Exempel 9

119

OM Window

- Mer användbart: en klocka i statusraden

```
function display_time() {
var d = new Date();
var h = d.getHours();
var m = d.getMinutes();
var s = d.getSeconds();
if (h == 0)
h = 00;
if (m < 10)
m = "0" + m;
if (s < 10)
s = "0" + s;
var time = h + ":" + m + ":" + s;
defaultStatus = time;
}
setInterval(display_time, 1000);
```

Exempel 10

120

OM Navigator

- Webbbläsaren som helhet
 - Version
 - Vilka dataformat den kan hantera
- Ett exempel

```
var browserMsg = "Information om webbläsaren:\n";
for(var propname in navigator) {
  browserMsg += propname + ": " + navigator[propname] + "\n";
}
alert(browserMsg);
```

121

OM Vänta lite nu...

- Iteration `for/in`
 - Hämta värden utifrån en sträng istället för en identifierare
 - Dynamiskt associera godtyckliga värden med godtyckliga strängar
 - Användbart när
 - namnen på objektets egenskaper inte är kända
 - När vi vill hämta alla värden med få rader kod
 - Egenskaper kan adderas dynamiskt
 - Någon mekanism måste finnas som hanterar detta
 - Objekt fungerar därför som associativa arrayer
 - Alla objekt har `toString()` definierad
 - Definierar hur objektet ska skrivas ut till en sträng

122

OM Navigator

- Ett exempel



- Tips
 - Använd `parseInt()` och `String.indexOf()`
 - Kan då extrahera exakt den information man vill ha

123

OM Screen

- Bildskärmen
 - Storlek
 - Antal färger

Några viktiga egenskaper

```
availHeight //tillgänglig höjd i pixels
availWidth //tillgänglig bredd i pixels
```

- Exempel

```
var maxHeight = window.screen.availHeight;
```

124

OM Location

- URL
 - För aktuellt visat dokument
 - Egenskapen är i sig en URL!



Några användbara egenskaper

```
href //full sökväg
protocol //associerat protokoll
search //extrahera info från frågesträngen
```

Några användbara metoder

```
reload //ladda om sidan
replace //byta ut dokument utan ny session
```

125

OM Location

- Exempel

```
window.location = "evilPage.html";
```

```
window.location.replace("hello.html");
```

Se upp med `window.navigate()` (gäller endast IE)

```
window.alert(location.href);
```

Exempel 11

126

OM History

- Fönstrets historia

- Har en array av tidigare besökta URL:er
- Elementen i arrayen är inte direkt åtkomliga att syna!

Metoder

```
back()      //föregående URL visas, (knappen BACK ungefär)
forward()   //framförvarande URL visas, (knappen FORWARD)
go()        //går till specifik URL i historiken
```

Egenskaper

```
length      //antal besökta sidor i historiken
```

- Nyttigt exempel i kurslitteraturen s217

127

OM Dokumentation av API

- Allt bör finnas dokumenterat
- Utnyttja som lexikon
- Viktigt!
 - Där man söker finner man
 - Uppfinn inte hjulet på nytt
 - Kompatibilitet
- Ni kommer att behöva utnyttja OM !

(Client-Side JavaScript Reference sid 541-678)

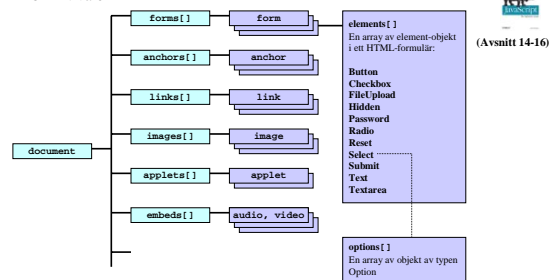
128

W3C DOM Nivå 0

129

DOM Nivå 0

DOM Nivå 0



130

DOM Nivå 0

- Standard?
 - Utgör ett historiskt arv från de svåra krigsåren
 - Samlingsnamn för alla tidigare vilda försök
 - Ingår numera som formaliserad del av DOM Nivå 1
 - Ska garantera fortsatt framtida stöd
- Stöd
 - Nära universalt stöd hos webbläsare
 - Inkompatibilitet finns dock

131

DOM Nivå 0 Dokumenthantering

- Inte en fullvärdig DOM
 - Ej fullständig kontroll över dokumentet
 - Ej fullständig åtkomst till dokumentet
- Endast grundläggande funktioner
- Elementen åtkomliga genom datasamlingar
- Åtkomst till vissa element
- Lägga till/ta bort/modifiera vissa element
- Tillstånd med kakor (cookies)

132



DOM Nivå 0

Händelsehantering

- Propagering
 - En händelse avser ett, och endast ett, mål
 - Endast målnoden kan därför svara på händelsen (Avsnitt 12.3, 14.1.5 och 19.1)
 - Detta medför en decentraliserad hantering

133



DOM Nivå 0

Händelsehantering

- Registrering
 - Genom att tilldela ett elements attribut en sträng med JavaScript-kod
 - Genom att en egenskap hos ett objekt tilldelas en funktion
 - Oavsett teknik så används följande hanterare:

```
onclick, ondblclick  
onhelp  
onkeydown, onkeyup, onkeypress  
onmousedown, onmouseup  
onmousemove  
onmouseover, onmouseout
```

Se bild 183

- Dessa definieras i klassen HTMLElement i gamla DOM
- HTMLElement utgjorde där superklass för alla element

134

DOM Nivå 0

Händelsehantering

- Registrering
 - Endast en hanterare för en särskild typ av händelse och för ett speciellt objekt
 - Om händelser från flera element ska generera samma respons måste samma hanterare registreras på alla dessa element!
 - Exempelvis om muspekaren förs över godtycklig tagg <p>

135



DOM Nivå 0

Händelsehantering

- Interaktion med händelser
 - Ingen representation av händelser
 - Ingen åtkomst till händelser
 - Ingen manipulation av händelser
 - Det går inte att generera egna händelser

136

DOM Nivå 0

Objekten

- Utvalda objekt kommer att tas upp
 - `document`
 - `form`
- Se upp med egenskaper och metoder specifika för olika webbläsare

137

document

DOM Nivå 0

Document

- Representerar det dokument som visas i fönstret
- Rot-objektet i DOM-hierarkin
- Information om dokumentets innehåll
- Dokumentåtkomst och hantering
- Tillstånd
- Händelser

(Avsnitt 14)

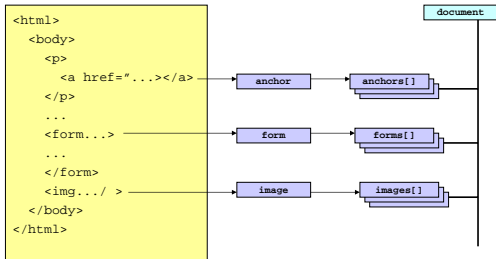


Se bild 185

138

DOM Nivå 0 Document

- Hur fungerar det?



139

DOM Nivå 0 Document

Några viktiga egenskaper

`forms[]`
`anchors[]`
`links[]`
`images[]`
`applets[]`
`embeds[]`

Andra egenskaper

`cookie` //för kakor
`lastModified` //senast uppdaterad
`linkColor` //obesökta länkar
`body` //taggen `<body>`
`title` //värdet i `<title>`

Några användbara metoder

`write()` //skriver strängar till dokumentet
`writeln()` //som `write()` fast med radbyte

140

DOM Nivå 0 Document

- Åtkomst
 - Namnge element och indexera egenskaperna

```
<form name="form1">
  <input type="text" name="x" value="" />
  <input type="button" name="send" value="Sänd" />
</form>
```

```
var form = document.forms[0];
form = document.forms.form1;
form = document.forms["form1"];
var button = document.forms.form1.elements["send"];
var v = document.forms.form1.elements["x"].getAttribute("value");
```

HJÄLP!

```
var v = document.all["x"].getAttribute("value"); //Endast IE4+
```

141



DOM Nivå 0 Document

- Dynamiskt innehåll

```
document.write("Nu skriver vi till dokumentet");
document.writeln("Nu skriver vi och byter rad");
document.write(msg, msg2) //argumenten "konkateneras"
```

- Om vi vill lägga till ett godtyckligt element då?

```
document.write("<div>Usch och fy då</div>");
```

- Kan bara skriva medan dokumentet tolkas!
 - Inifrån händelsehanterare medför att dokumentet skrivs över
 - Men i kombination med `open()` och `close()` kan man generera helt nya dokument istället (fortfarande inbäddad HTML-kod dock)!

142

DOM Nivå 0 Document

- Begränsad hantering
 - Element utanför strukturen oåtkomliga
 - `<div>`
 - ``
 - etc.
 - Lägga in nya element
 - Endast `document.write()`, en mycket tveksam metod!
 - HTML-kod skulle därmed skrivas direkt i JavaScript-filen
 - Dessutom kan vi inte skriva efter att dokumentet tolkats klart

143

DOM Nivå 0 Document

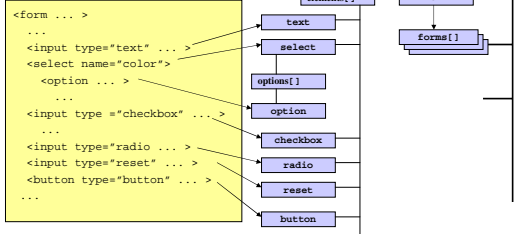
- Händelsehanterare direkt i HTML-filen
 - Tilldela ett elements attribut en sträng med JavaScript-kod
- Händelsehanterare i JavaScript-fil
 - En egenskap hos ett objekt tilldelas en funktion
- I båda fallen gäller att:
 - En funktion tilldelas en egenskap hos ett element i dokumentet

144

DOM Nivå 0

Form

- Hur fungerar det?



151

DOM Nivå 0

Form

```

var form = document.forms["test"];

function addHandlers(elements) {
  var check = form.elements["sv"];
  var radio = form.elements["browser"];
  check.onclick = function() { effect(radio); }
  var reset = form.elements["reset"];
  reset.onclick = function() { effect(reset); }
  var send = form.elements["send"];
  send.onclick = function() { effect(send); }
}
  
```

152

DOM Nivå 0

Form

```

function effect(toChange) {
  if (toChange.type == "radio")
    toChange.checked = true;
  else if (toChange.type == "reset")
    form.reset();
  else if (toChange.type == "button") {
    form.action = "exempell4b.html";
    form.method = "post";
    form.submit();
  }
}

addHandlers();
  
```

Exempel 14

153

DOM Nivå 0

Cookie

- Data
 - Lagras i webbläsaren
 - Associerad med en webbsida
 - Tjänar som "minne" och tillstånd
- Ej ett objekt i hierarkin
- En sträng
- Kakor utgörs av name=value

```
document.cookie = "course=IP3"
```

- Kakan försvinner med sessionen



(Avsnitt 16)

154

DOM Nivå 0

Cookie

- Sätta utgångsdatum

```
document.cookie = "name=value"; expires date
```

- Argumentet måste i detta fall vara en sträng i formatet som `Date.toGMTString()` returnerar:

```

var nextYear = new Date();
var yearLater = nextYear.getFullYear() + 1;
nextYear.setFullYear(yearLater);
var expires = nextYear.toGMTString();

document.cookie = "course=Pierre"; expires = "" + expires;
  
```

155

DOM Nivå 0

Cookie

- Man kan lägga till fler kakor

```
document.cookie = "course=IP3"
```

```
document.cookie = "teacher=Pierre"
```

- Lagras dock i samma sträng!
- Som en lista av namn/värde-par separerade av kommatecken
- Använd följande metoder för att extrahera informationen `String.indexOf()`, `String.substring()`, `String.split()`

156

DOM Nivå 0 Slutsatser

- Begränsad dokumenthantering
- Begränsad händelsehantering
- DOM Nivå 0 räcker dock inte till
- Viktig att känna till dock
- Det finns några användbara aspekter
- Vi ska använda DOM Nivå 2!

157

DOM Nivå 0 Dokumentation av API

- Allt bör finnas dokumenterat
- Utnyttja som lexikon
- Viktigt!
 - Där man söker finner man
 - Uppfinn inte hjulet på nytt
 - Kompatibilitet

(Client-Side JavaScript Reference sid 541-678)

158

W3C DOM Nivå 2

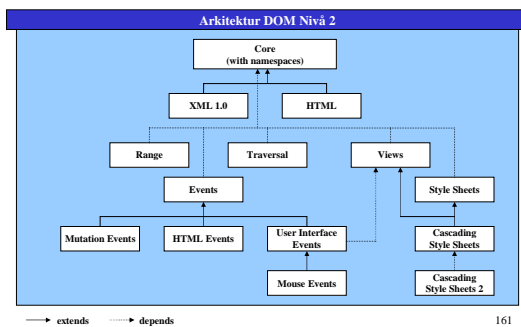
159

DOM Nivå 2

- Senaste standarden
- Bygger vidare på DOM Nivå 1
- Generell (HTML, XML etc.)
- Olika nivåer
- Modulbaserat
- Varje modul är ett API
- Modulerna bygger på gränssnitt (*interface*)
- Vid implementering skapas en komplex hierarki med arv

160

DOM Nivå 2 Arkitektur



161

DOM Nivå 2

- Grundläggande dokumenthantering
 - Core
- Utökad hantering för HTML
 - HTML
- Händelsehantering
 - Events
 - User Interface Events,
 - Mouse Events
- Hantering av CSS
 - Style sheets
 - CSS
 - CSS2

162

DOM Nivå 2

Dokumenthantering

- Dokument utgör en hierarkisk struktur
- Representeras i en trädstruktur
- Metafor: Familjeträdet
 - Förälder (*parent*)
 - Barn (*child*)
 - Syskon (*sibling*)
 - Avkomma (*descendant*)
 - Förfäder (*ancestor*)

163

DOM Nivå 2

Dokumenthantering

- Trädstruktur med olika typer av noder
 - Element eller taggar som `<body>`
 - Textsträngar
 - Kommentarer
- Metoder och egenskaper för att
 - Stega igenom trädet
 - Identifiera godtyckliga noder och element
 - Extrahera information
 - Modifiera struktur och innehåll
- Stöd för olika dokument samt namespaces

164

DOM Nivå 2

Dokumenthantering

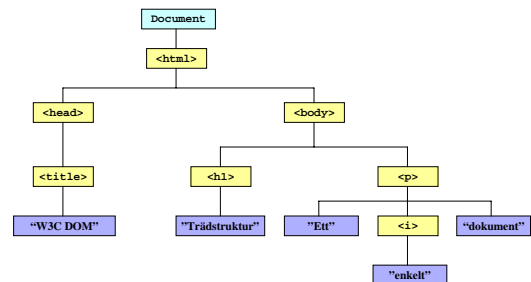
```
<html>
<head>
  <title>
    W3C DOM
  </title>
</head>
<body>
  <h1>
    Trädstruktur
  </h1>
  <p>
    Ett <i>enkelt</i> dokument
  </p>
</body>
</html>
```

Ej XHTML!
Endast exempel!

165

DOM Nivå 2

Dokumenthantering



166

DOM Nivå 2

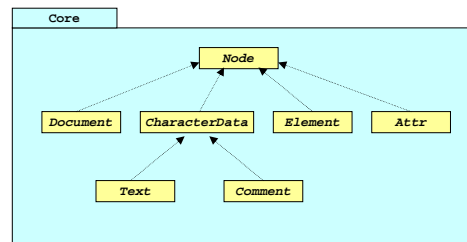
Dokumenthantering

- Ett träd i DOM består av
 - Noder, alltså objekt av typen *Node*
- Varje träd har en rot
- Resten av trädet byggs upp med
 - Objekt av typen *Element*
 - Objekt av typen *Text*
 - Objekt av typen *Comment*
- *Element* representerar
 - Taggar
 - Har attribut vilka representeras av typen *Attr*
- *Text* representerar
 - Textsträngar, dvs information
- *Comment* representerar
 - Kommentarer

167

DOM Nivå 2

Core




168

Node

DOM Nivå 2
Core

- En abstrakt nod



(W3C DOM Reference sid 812)

Några användbara egenskaper

```

nodeType //konstant som definierar nodens typ
nodeName //namnet (för ex elementnode, taggens namn)
nodeValue //värdet (för ex. textnode, deras innehåll)
childNodes[] //alla barn (inneslutande element)
firstChild //första inneslutande elementet
lastChild //sista inneslutande elementet
previousSibling //föregående syskon
nextSibling //efterföljande syskon

```

169

DOM Nivå 2
Core

Några användbara metoder

```

appendChild() //lägg till nod i trädet
removeChild() //ta bort nod i trädet
replaceChild() //ersätt nod i trädet
insertBefore() //lägg till nod före annan nod

```

170

DOM Nivå 2
Core

- nodeType
 - Ta reda på typ av nod

Interface	nodeType constant	nodeType value
<i>Element</i>	Node.ELEMENT_NODE	1
<i>Text</i>	Node.TEXT_NODE	3
<i>Document</i>	Node.DOCUMENT_NODE	9
...

- Om noden är ett objekt av typen *Element*
 - Tillgång till egenskaper och metoder hos både *Node* och *Element*
 - Motsvarande gäller för övriga
 - Interfacen ärver från varandra

171

DOM Nivå 2
Core

- Så här har W3C tänkt

```

var node = document.childNodes[5];
if (node.nodeType == Node.ELEMENT_NODE) {...}

```

- IE 6 stödjer tyvärr inte detta!!!
 - Referera till det numeriska värdet istället
 - Oacceptabelt att behöva göra detta
 - Att definiera konstanter är god programmering
 - Det är därför W3C gjort det!

```

var node = document.childNodes[5];
if (node.nodeType == 1 ) //Node.ELEMENT_NODE


```

172

Document

DOM Nivå 2
Core

- Rotnoden för ett dokumentets träd



(W3C DOM Reference sid 720)

Några användbara egenskaper

```

docType
documentElement
styleSheets
implementation //för att skapa ex ett CSS
styleSheets //array av CSS:er

```

Några användbara metoder

```

createElement() //skapa nytt element
createAttribute() //skapa ett attribut
createTextNode() //skapa ett textelement
createEvent() //tas upp längre fram...
getElementById() //element med visst id
getElementsByTagName() //alla element av viss tagg

```

173

DOM Nivå 2
Core

- docType
 - Refererar till en dokumentdefinition
 - I HTML representeras detta av

```
<!DOCTYPE html PUBLIC ... >
```
- documentElement
 - Refererar till ett objekt av typen *Element*
 - Representerar rotelementet
 - I HTML-dokument utgörs detta av taggen `<html>`
 - Här kan attribut som `xmlns` finnas
 - Kan finnas andra barn som kommentarer

174


Element

DOM Nivå 2
Core

- Representerar taggar i dokumentet
 - Exempel i HTML
<html>, <body>, <p> och <div>
 - Både starttagg och sluttagg inkluderas

En användbar egenskap

tagName //ex. P för <p>
//observera versalen
//motsvarar nodeName i interfacet Node



(W3C DOM Reference sid 744)

175

DOM Nivå 2
Core

Några användbara metoder

getAttribute() //returnerar ett attribut
getAttributeNode()
getElementsByTagName() //returnerar array av element
//som matchar ett tagg-namn
removeAttribute()
setAttribute()
setAttributeNode()

176


Attr

DOM Nivå 2
Core

- Motsvarar attribut i element
 - Exempel i HTML
src i taggen
href i taggen <a>
- Attribut kan också
 - Utforskas
 - Modifieras
 - Läggas till
 - Tas bort
- Attribut är inte direkt en del av dokumentträdet på samma sätt som textnoder och element är

Några användbara egenskaper

name //ex. href
value //ex "http://www.dsv.su.se"



(W3C DOM Reference sid 692)

177

CharacterData

DOM Nivå 2
Core


- Allmän funktionalitet för
 - Textnoder
 - Kommentarer

En användbar egenskap

data //texten

Några användbara metoder

appendData()
deleteData()
insertData()
replaceData()



(W3C DOM Reference sid 693)

178


Text

DOM Nivå 2
Core

- Representerar ren text
 - Exempel i HTML
<p>Detta är en textsträng</p>

En användbar metod?

splitText()




(W3C DOM Reference sid 846)

179

Comment

DOM Nivå 2
Core

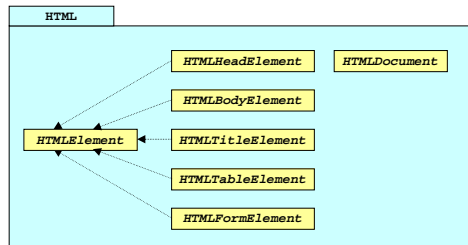
- Representerar kommentarer
 - Exempel i HTML
<!--detta är en kommentar i HTML-->
<!-- En kommentar med flera rader
<p>Bortkommenterat</p>
<p>Detta också</p> -->
- Åtkomst till texten
 - Genom egenskapen data i CharacterData
 - Genom egenskapen nodeValue i Node



(W3C DOM Reference sid 697)

180

DOM Nivå 2 HTML



181

DOM Nivå 2 HTML

- Nya gränssnitt
 - Somliga ärver från gränssnitt i Core
 - Utökas med nya egenskaper
 - Utökas med nya metoder
 - HTML-specifika funktioner

182

DOM Nivå 2 HTML

- Basgränssnittet för alla HTML-element
 - Implementeras av alla HTML-element
 - Definierar egenskaper som representerar attribut som alla element har
- Element som accepterar fler attribut
 - har egna gränssnitt som implementerar *HTMLElement*
 - utökade egenskaper för fler attribut

```
<span> //finns fler
```

```
HTMLHeadElement
HTMLBodyElement
HTMLTitleElement
...
```

(W3C DOM Reference sid 775)

183

DOM Nivå 2 HTML

Egenskaper (motsvarar attribut)

```
style
className
dir
id
lang
title
```

- Händelsehanterare
 - De som definieras i *HTMLElement* i DOM Nivå 0

Se bild 133

(Client-Side JavaScript Reference sid 579)

184

DOM Nivå 2 HTML

- Ärver från *Document*
- Definierar HTML-specifika
 - Egenskaper
 - Metoder
- Möjliggör kompatibilitet med DOM Nivå 0
- Utgör alltså en formalisering av
 - Document* i gamla DOM
- Har inte behållit alla egenskaper
- Somliga har flyttats till andra gränssnitt

(W3C DOM Reference sid 771)

(Client-Side JavaScript Reference sid 551)

185

DOM Nivå 2 HTML

- Utgör roten i trädet för ett HTML-dokument

Några viktiga egenskaper

```
forms[]
anchors[]
links[]
images[]
applets[]
```

Andra egenskaper

```
body //elementet <body>
cookie
lastModified
linkColor
title
navbar
```

Några användbara metoder

```
write() //skriver strängar till dokumentet
writeln() //som write() fast med radbyte
```

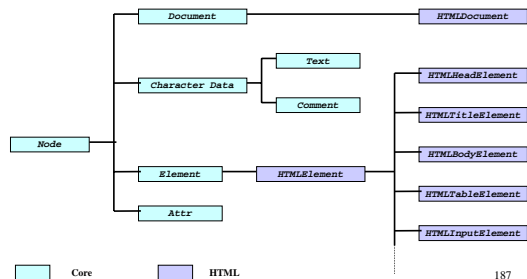
Se bild 137

186

DOM Nivå 2

Översikt

Ett partiellt klassdiagram av Core och HTML



187

DOM Nivå 2

Att arbeta med Core och HTML

- Stega igenom dokument
 - Hitta element
 - Addera innehåll
 - Modifiera dokument
-
- Dokumentet måste vara helt inläst i webbläsaren
 - Detta är ett generellt krav

188

DOM Nivå 2

Stega igenom dokument

- Inspektera och räkna noderna

```
<html>
<head>
  <title>W3C DOM</title>
</head>
<body>
  <h2>Räkna antalet taggar</h2>
  <p>
    Vi stegar igenom dokumentet och räknar antalet taggar.
  </p>
  <script type="text/javascript" src="exempel15.js"></script>
</body>
</html>
```

189

DOM Nivå 2

Stega igenom dokument

- Inspektera och räkna noderna

```
function countTags(n) { //n är av typen Node
  var noOfTags = 0;
  if (n.nodeType == 1 /*Node.ELEMENT_NODE*/) { //n är ett element
    noOfTags++;
  }
  var children = n.childNodes();
  for (var i = 0; i < children.length(); i++) { //iterera genom alla barn
    noOfTags += countTags(children[i]); //och deras barn
  }
  return noOfTags;
}
countTags(document); //dokumentets rot
```

Exempel 15

190

DOM Nivå 2

Stega igenom dokument

- Räkna antalet tecken

```
<html>
<head>
  <title>W3C DOM</title>
</head>
<body>
  <h2>Räkna antalet tecken</h2>
  <p>
    Vi stegar igenom dokumentet och räknar antalet
    tecken i textelement innanför taggen body.
  </p>
  <p>
    Taggarna p är syskon.
    <span>Sista p-taggen har detta barn</span>
  </p>
  <script type="text/javascript" src="exempel16.js"></script>
</body>
</html>
```

191

DOM Nivå 2

Stega igenom dokument

```
function countCharacters(n) {
  //Om n är ett textelement
  if (n.nodeType == 3 /*Node.TEXT_NODE*/) {
    return n.length;
  }
  var noOfChars = 0;
  //Börja vid första barnet
  //Kör tills inga barn längre hittas
  //Stega igenom varje syskon
  for (var c = n.firstChild; c != null; c = c.nextSibling) {
    noOfChars += countCharacters(m);
  }
  return noOfChars;
}
countCharacters(document.body);
```

Exempel 16

192

DOM Nivå 2

Hitta element

- Returnera ett unikt element
 - `getElementById()` hos *Document*
- Returnera alla element av en viss typ
 - `getElementsByTagName()` hos *Document*
- Returnerar en grupp av element
 - `getElementsByTagName()` hos *Element*
Returnerar alla element av en viss typ inom ett element

193

DOM Nivå 2

Hitta element

- I föregående exempel använde vi
 - `document.body` (egenskap i *HTMLDocument*)
- Vi kan också använda

```
document.getElementsByTagName("body")[0];
```

 - Returnerar en array av alla element som representerar taggen `<body>`
 - Vi väljer det första i denna array :))

194

DOM Nivå 2

Hitta element

- Andra exempel
 - Returnera alla element som representerar taggen `<table>`

```
document.getElementsByTagName("table");
```
 - Returnera alla element

```
document.getElementsByTagName("*");
```

För IE5 och IE 5.5 får man skriva

```
document.all[];
```

195

DOM Nivå 2

Hitta element

- Mycket användbart och vanligt
 - Returnera elementet utifrån en identifierare
 - Taggens attribut `id` används och ges ett unikt värde

```
<p id="x1">
  Nu har vi gett denna tagg en identifierare.
</p>
```
 - Vi kan hämta elementet genom

```
var element = document.getElementById("x1");
```
 - Mer kraftfullt och mycket enklare än DOM Nivå 0!

196

DOM Nivå 2

Hitta element

- Arbeta med en grupp element inom ett element

```
<html>
<head>
  <title>W3C DOM</title>
</head>
<body>
  <h2>Hämta en grupp av element</h2>
  <p>Detta är en grupp.</p>
  <p id="x1">
    <span>Nu har vi gett denna grupp en identifierare.</span>
    <span>Detta är den grupp vi vill hämta elementen från.</span>
  </p>
  <script type="text/javascript" src="exempel17.js"></script>
</body>
</html>
```

197

DOM Nivå 2

Hitta element

- Arbeta med en grupp element inom ett element

```
function findGroup(id) {
  var element = document.getElementById(id);
  var group = element.getElementsByTagName("span");
  return group.length;
}

window.alert("Antal element i gruppen: " + findGroup("x1"));
```

Exempel 17

198

DOM Nivå 2

Addera innehåll

- Addera ett nytt element

```
<html>
<head>
  <title>W3C DOM</title>
</head>
<body>
  <h2>Lägg till ett element</h2>
  <p id="x2">
    Vi lägger till ett nytt stycke innanför detta.
    Stycket får sitt attribut "id" satt.
  </p>
  <script type="text/javascript" src="exempel18.js"></script>
</body>
</html>
```

199

DOM Nivå 2

Addera innehåll

```
var no = 0;
function add() {
  var parent = document.getElementById("x2"); //föräldern
  var element = document.createElement("p"); //elementet
  element.setAttribute("id", "p" + ++no); //attributet
  var msg = "Här är vårt nya element med en text. "; //texten
  msg += "Värdet för attributet id är: ";
  msg += element.getAttribute("id"); //textnoden
  var data = document.createTextNode(msg); //addera text
  element.appendChild(data); //addera element
  parent.appendChild(element);
}
add();
```

Exempel 18

- Bortse från händelsehanteringen tills vidare!

200

DOM Nivå 2

Modifiera dokument

- Byta plats på elementen

```
<html>
<head>
  <title>W3C DOM</title>
</head>
<body>
  <h2>Byta plats på element</h2>
  <p>Stycke 1</p>
  <p>Stycke 2</p>
  <p>Stycke 3</p>
  <script type="text/javascript" src="exempel19.js"></script>
</body>
</html>
```

201

DOM Nivå 2

Modifiera dokument

```
function reverse(n) {
  var kids = n.childNodes; //alla barn i <body>
  var noOfKids = kids.length; //antal barn
  for (var i = (noOfKids - 1); i >= 0; i--) { //iterera baklänges
    var child = n.removeChild(kids[i]); //ta bort
    n.appendChild(child); //lägg till sist
  }
}
reverse(document.body);
```

Exempel 19

- Bortse från händelsehanteringen tills vidare!

202

DOM Nivå 2

Modifiera dokument

- Ta bort element, uppdatera text

```
<html>
<head>
  <title>W3C DOM</title>
</head>
<body>
  <h2>Ta bort element och uppdatera text</h2>
  <p id="p1">Detta stycke ska vi ta bort</p>
  <p id="p2">Denna text ska vi byta ut</p>
  <script type="text/javascript" src="exempel20.js"></script>
</body>
</html>
```

203

DOM Nivå 2

Modifiera dokument

```
function update() {
  var remove = document.getElementById("p1"); //första taggen p
  var e2 = document.getElementById("p2"); //andra taggen p
  var update = e2.firstChild; //textelementet
  var data = "Nu har vi uppdaterat texten."; //nya texten
  document.body.removeChild(remove); //ta bort första
  update.replaceData(0, update.data.length-1, data); //uppdatera texten
}
update();
```

Exempel 20

- Bortse från händelsehanteringen tills vidare!

204



DOM Nivå 2

Händelsehantering

- Propagering i tre faser
 - *Capturing*
 - Händelser propageras från dokumentobjektet
 - Färdas genom trädets objekt ner till målnoden
 - Om förfäder har hanterare registrerade körs dessa
 - Hos målnoden själv
 - Registrerade hanterare exekverar (vilket motsvarar DOM 0)
 - *Bubbling*
 - Händelsen propageras upp igen
 - Från målet
 - Genom dokumenthierarkin
 - Upp till dokumentobjektet
- Detta ger oss centraliserad hantering

205



DOM Nivå 2

Händelsehantering

- Registrering
 - Specifika metoder för att registrera hanterare
 - Flera hanterare kan registreras för en viss typ av händelse och objekt
 - Händelsehanterare kan registreras på olika nivåer
 - Exempelvis registrering av en hanterare i dokumentobjektet och hantera händelserna under faserna *capturing* eller *bubbling*

206



DOM Nivå 2

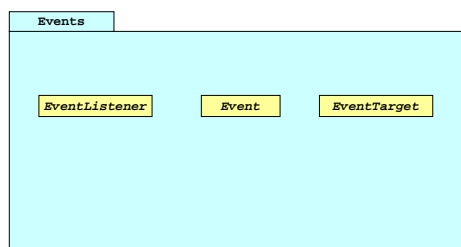
Händelsehantering

- Interaktion med händelser
 - Särskilda objekt representerar händelser
 - JavaScript kan få åtkomst till dessa objekt
 - JavaScript kan skapa sådana objekt
 - Jämför med undantag i Java

207

DOM Nivå 2

Events



208

Event

DOM Nivå 2

Events

- Information om en händelse

(W3C DOM Reference sid 759)

Några användbara egenskaper

```
target //målnoden, dvs den som genererade händelsen
type //registrerade hanterarens namn, ex "click"
```

Några användbara metoder

```
initEvent() //initierar ett eget skapat händelseobjekt
stopPropagation() //stoppar propageringen
```

209

EventListener

DOM Nivå 2

Events

- Definierar strukturen för en händelsehanterare

(W3C DOM Reference sid 762)

Metoder

```
handleEvent() //funktionen som ska utföras
```

- Vilken funktion som helst dock!
- Kan ta ett argument som utgör händelseobjektet som skickas med
- Behöver inte ens ta detta argument
- Inte som i språk som Java med andra ord

210

DOM Nivå 2

Events

- Hanterar registrering av händelsehanterare



(W3C DOM Reference sid 763)

Metoder

```
addEventListener() //registrera hanterare
dispatchEvent() //sätter en mÅlnod f r ett
                //h ndelseobjekt som skapats
                //med document.createEvent()
                //initierats med
                //Event.initEvent()
removeEventListener() //Avl gsnar en hanterare
```

211

DOM Niv  2

Events

addEventListener(type, listener, useCapture)

- type: str ng med namnet f r typen av h ndelse
 - "click", "mouseover", "mouseout" etc.
- listener: hanteraren
 - Funktionen som ska utf ras
- useCapture: Ett booleskt v rde
 - Om sant, f ngas h ndelserna under *capturing*
 - Om falskt, f ngas h ndelserna om dom sker direkt  ver elementet eller  ver en avkomma och d rmed "bubblar" upp till elementet

212

DOM Niv  2

Arbeta med Events

- Registrering av en h ndelsehanterare

```
<html>
...
<div id="x1">
  Vi vill registrera en h ndelsehanterare
  f r denna tagg.
</div>
...
<form id="form1" ...> //och en h r
...
</form>
</html>
```

213

DOM Niv  2

Arbeta med Events

- En specifik h ndelse till ett specifikt element

```
var myForm = document.getElementById("form1");
myForm.addEventListener("submit",
  function() { validate(); }, false);
```

- Alla h ndelser av en viss typ inom ett element

```
var mydiv = document.getElementById("x1");
mydiv.addEventListener("mousedown",
  handleMouseDown, true);
```

- validate() och handleMouseDown m ste definieras

214

DOM Niv  2

Events och kompatibilit t

- Tyv rr kan vi inte festa  nnu...
- Microsoft har fortfarande en egen uppfattning om
 - Hur en h ndelsehanterare ska registreras
 - Hur en h ndelse ska representeras
 - Vart h ndelsen skickas

215

DOM Niv  2

Events och kompatibilit t

- H ndelseobjektet
 - Skickas inte med till hanteraren i IE, d v s funktionen
 - Finns ist llet som egenskap hos Window
 - Funktionen kan d  referera till window.event
 - N ja, inget krav att f rv ntade argument verkligen skickas...
- IE St djer *bubbling*, men inte *capturing*
 - Implementerar inte stopPropagation()
 - Ist llet window.event.cancelBubble = true;
- H ndelseobjektet allm nt
 - Varken NN eller IE f ljer standarden helt m a p
 - Metoder
 - Egenskaper
 - Egenskapen type st ds dock av b da

216

DOM Nivå 2

Events och kompatibilitet

- Mer störande för vår del
 - IE Implementerar inte `addEventListener()`, `removeEventListener()`
 - Istället implementeras `attachEvent()`, `detachEvent()`
- Metoderna tar bara två argument
 - Det booleska värdet utblir, har ju med *capturing* att göra
- Vidare...
 - Prefixet "on" måste ingå i namnet för typen av händelse
 - D v s "onclick", "onmouseover".
- Nu måste vi skriva kod som kringgår detta
Tack Microsoft!

217

DOM Nivå 2

Events och kompatibilitet

```
ie = (document.all) ? true:false; // IE4+
dom = ((document.getElementById) && (!ie)) ? true:false; // Mozilla 0.9+

/*
 * Registers an event listener for the given type of event on
 * an object with the given identifier. The given function
 * acts as the listener and will be called to respond when
 * events arise.
 */
function setEventById(id, ev, fu) {
  if(dom) {
    document.getElementById(id).addEventListener(ev, fu, false);
  }
  if(ie) {
    document.getElementById(id).attachEvent('on' + ev, fu);
  }
}
```

Se bild 220

218

DOM Nivå 2

Events och kompatibilitet

- Validering av formulär

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Händelsehantering</title>
</head>
<body>
<h2>Kontroll med återkoppling</h2>
<form action="exempel21b.html" method="post" id="form1">
<span>Namn:<input type="text" value="" id="text" name="Namn" /></span>
<input type="text" value="" id="text" name="Namn" /><input type="button" id="send">Skicka</button>
</form>
<script type="text/javascript" src="exempel21.js"></script>
</body>
</html>
```

219

DOM Nivå 2

Events och kompatibilitet

- Implementering av `validate()`

```
function validate() {
  var form = document.getElementById('form1');
  var input = document.getElementById('text');

  if ((input.getAttribute("value") == "")) {
    window.alert("Fyll i följande fält:\n\n" +
      input.getAttribute("name"));
  } else {
    form.submit();
  }
}
```

220

DOM Nivå 2

Events och kompatibilitet

- Så anropar vi vår `setEventById()`

Se bild 217

```
setEventById('send', 'click', validate);
```

Exempel 21

- Värre än så blir det inte ☹
- En aspekt att fundera över dock
 - Hur löser vi *capturing*?
 - Argumentet sätts alltid till "false" nu
 - Om vi vill ha *capturing* istället för elementspecifik hantering?

221

DOM Nivå 2

Events och kompatibilitet

- Manipulera händelseobjektet

```
...
function alertUser(e) {
  if (ie) //i IE lagras händelseobjektet hos window
    window.alert("Händelsetyp: " + window.event.type);
  else if (dom) //enlig DOM 2 ska det skickas till funktionen
    window.alert("Händelsetyp: " + e.type);
}

setEventById('x', 'click', alertUser);
```

Exempel 22

222

DOM Nivå 2

Interaktion med CSS

- Deklarera ny stilregel
- Byta mellan existerande regler
- Skapa en CSS-fil
- Länka ett HTML-dokument till en CSS-fil?
- Stega mellan flera CSS-filer



(Avsnitt 18)

229

DOM Nivå 2

StyleSheets

StyleSheets

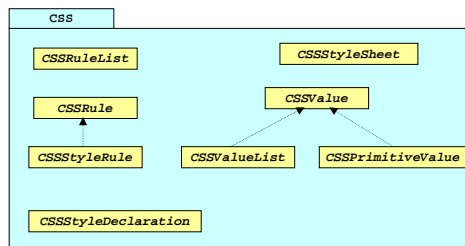
StyleSheet

StyleSheetList

230

DOM Nivå 2

CSS



231

DOM Nivå 2

CSS2

CSS2

CSS2Properties

232

DOM Nivå 2

Deklarera ny stilregel

- Vi arbetar nu inte mot en existerande CSS-fil 🚫
 - Vi skapar istället en regel direkt hos elementet
 - Detta sker så att säga on-the-fly, eller inline

```
<html>
...
<p id="text">En text med tråkigt teckensnitt</p>
...
</html>
```

```
var element = document.getElementById("text");
element.style.fontFamily = "sans-serif";
element.style.setProperty("font-family", "sans-serif", "");
```

233

DOM Nivå 2

Byta mellan existerande regler

- Det finns en CSS-fil med deklarerade regler 😊
 - Vi kopplar elementet till en av dessa

```
.out {}
.new_spacing {
  letter-spacing: 2px;
}

.new_color {
  color: #0000FF;
  background: #FFFFFF;
}
```

234

DOM Nivå 2

Byta mellan existerande regler

```
<html>
<head>
  <title>JavaScript och CSS</title>
  <link href="exempel26.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <h2>Stilsätta med regler definierade i en CSS-fil</h2>
  <p id="x1" class="out">
    Ändra avstånd mellan bokstäver.
  </p>
  <p id="x2" class="out">
    Ändra färg på texten.
  </p>
  <script type="text/javascript" src="exempel26.js"></script>
</body>
</html>
```

235

DOM Nivå 2

Byta mellan existerande regler

```
function changeStyleOut() {
  document.getElementById('x1').className = 'out';
}
function changeStyleOver() {
  document.getElementById('x1').className = 'new_spacing';
}
function changeColorOut() {
  document.getElementById('x2').className = 'out';
}
function changeColorOver() {
  document.getElementById('x2').className = 'new_color';
}
setEventById('x1', 'mouseover', changeStyleOver);
setEventById('x1', 'mouseout', changeStyleOut);
setEventById('x2', 'mouseover', changeColorOver);
setEventById('x2', 'mouseout', changeColorOut);
```

Exempel 26

236

DOM Nivå 2

Byta mellan existerande regler

- Ett exempel till

```
div.animate1 {
  color: #000000;
  background-color: white;
}
div.animate2 {
  color: #000000;
  background-color: yellow;
}
div.animate3 {
  color: #000000;
  background-color: orange;
}
div.animate4 {
  color: #000000;
  background-color: red;
}
```

237

DOM Nivå 2

Byta mellan existerande regler

```
<html>
<head>
  <title>JavaScript och CSS</title>
  <link href="exempel27.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <h2>En animation</h2>
  <div id="x1">
    <h3>Webbservern är under attack</h3>
  </div>
  <script type="text/javascript" src="exempel27.js"></script>
</body>
</html>
```

238

DOM Nivå 2

Byta mellan existerande regler

```
var no = 1;

function animate(element){
  element.className = "animate" + no;
  (no > 4) ? no = 1 : no++;
}

var e = document.getElementById("x1");
setInterval('animate(e)', 100);
```

Exempel 27

239

DOM Nivå 2

Skapa en CSS-fil

- Man kan skapa en ny fil

```
var sheet = document.implementation.createCSSStyleSheet();
```

– Detta skapar ett objekt av typen *CSSStyleSheet*

- Därefter kan man lägga till och ta bort regler

```
sheet.insertRule("p { font-size: 0.7em; }", 0);
sheet.deleteRule(0);
```

- Kuggfråga - var skapas/sparas filen?

240

DOM Nivå 2

Länka ett HTML-dokument till en CSS-fil?

- Om vi vill använda vår CSS-fil?
 - Då måste vi länka in det i HTML-filen!
 - Tyvärr finns det inget sätt ännu!
 - Kanske när DOM Nivå 3 kommer?

241

DOM Nivå 2

Åtkomst till existerande CSS-filer

- CSS:er som redan från början är länkade i HTML-filen
 - Är åtkomliga!
 - Dokumentet har en lista av dem
 - Varje CSS har en lista med regler

```
var sheet = document.styleSheets[0];  
var rule = sheet.cssRules[0];
```

- En regel består av en definition `div.animate1`
- Samt en deklaration med alla attribut och värden exklusive `{}`

242

DOM Nivå 2

Länka ett HTML-dokument till en CSS-fil?

- Rent hypotetiskt...

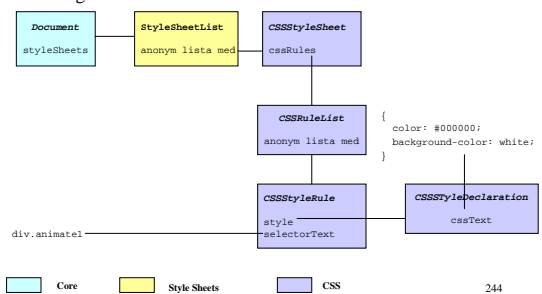
```
function animate(element){  
  var sheet = document.styleSheets.item(0); //styleSheets[0]  
  var rule, name;  
  
  for(var i = 0; i < sheet.cssRules.length; i++) {  
    rule = sheet.cssRules.item(i); //cssRules[0]  
    name = rule.style.split(".", 2);  
    element.className = name;  
  }  
}  
animate(document.getElementById("animate"));
```

243

DOM Nivå 2

Hur fungerar det?

- Hur fungerar det?



244

DOM W3C

Dokumentation av API:er

- Allt bör finnas dokumenterat
- Utnyttja som lexikon
- Viktigt!
 - Där man söker finner man
 - Uppfinn inte hjulet på nytt
 - Kompatibilitet



(W3C DOM Reference sid 685-853)

245

Tips

Felsökning

- Om scripten inte fungerar
 - IE: Felmeddelanden
 - NN: Tools->Web Development->JavaScript Console
- Vad kan man tänka på
 - Placering av `<script></script>`
 - Rätt sökväg och filnamn?
 - Syntaxfel: ' och '"', semikolon, parenteser och måsvingar `{}`
 - Logiska fel
 - Inkompatibilitet mellan webbläsare?
 - Skiftlägeskänsligt och otypat!
 - Opererar jag på rätt objekt, med rätt metod och korrekta argument?
 - Är mina funktioner deklarerade och anropas?
 - Sker anropen till funktioner/metoder med rätt namn

246

Länkar JavaScript

- ECMA
<http://www.ecma-international.org>
- ECMAScript 3
<http://www.ecma-international.org/publications/standards/Ecma-327.htm>
- Netscape, JavaScript 1.5 Reference
<http://devedge.netscape.com/library/manuals/2000/javascript/1.5/reference/>
- Microsoft, JScript/EcmaScript3
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/jscript7/html/jsgrpecmafeatures.asp>
- JavaScript Tutorial
<http://www.howtocreate.co.uk/tutorials/index.php>
- JavaScript Kit
<http://wsabstract.com/>

247

Länkar DOM

- W3C
<http://www.w3.org>
- W3C DOM
<http://www.w3.org/DOM/>
- W3C DOM Nivå 2 specifikation
<http://www.w3.org/TR/1999/CR-DOM-Level-2-19991210/>

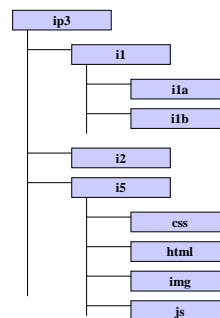
248

Länkar Övrigt

- Olika tutorials och lektstugor
<http://www.w3schools.com>
- Fler på kurshemsidan

249

Uppgifter Katalogstruktur



250