

10 World-Wide Web

10.5 Mobiler Code	2
10.5.1 Eingebetteter Code	3
10.5.2 JavaScript - Teil 1	8
10.5.3 Document Object Model	13
10.5.4 JavaScript - Teil 2	33
10.5.5 Applets etc.	44
10.6 AJAX	63
(wird fortgesetzt)	
Zusammenfassung	69

10.5 Mobiler Code

Jenseits einfacher HTML-Formulare hätte man gern
komplexe GUIs mit *klientenseitig* ausgeführtem Code !

Ansatz: nicht nur HTML, auch HTML+Code kann vom Server
bezogen und im Browser-Kontext interpretiert werden:
mobiler Code (Schlagwort: „dynamisches HTML“)

Zwei Techniken für das Herunterladen von mobilem Code:

- eingebetteter Code: `<script type=...> </script>`
- nachzuladender Code: `<script src=...>` oder `<object ...>`

Vorsicht vor Trojanischen Pferden !

10.5.1 Eingebetteter Code

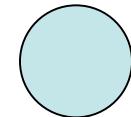
HTML-Elemente `script/noscript` für eingebetteten Code:

```
<html>
.....
<script type="text/vbscript">
    .....          (VBScript Code)
</script>
.....
<noscript><br>Cannot execute VBScript!</noscript>
```



Wird ausgegeben, wenn der Browser die gewünschte Skriptsprache nicht unterstützt.

```
<html>
<head>
<META http-equiv="Content-Script-Type" content="text/vbscript">
.....
</head>                                Gilt für gesamtes Dokument.
                                         Voreinstellung ist "text/javascript"
<body>
.....
<script type="text/javascript">
    .....                               ( JavaScript Code )
</script>
.....
<script>
    .....                               ( VBScript Code )
</script>
.....
```



! Ein HTML-Verweis (*Hyperlink, A element, tag <a>*) kann sich auf Skript-Code beziehen:

```
<html>
.....
<br>
Unsere Geschäftsbedingungen finden Sie
<a href="http://www.dot.com/agb.html">hier.</a>
<br><br>
Unsere Coldline erreichen Sie
<a href="mailto:info@dot.com">hier.</a>
Wenn Sie Ihren Namen eingeben wollen -
<a href="javascript: var name = prompt('Name:', '');
if(name=='') ..... ">hier.</a>
.....
</html>
```

Skript-Code kann auf eine Vielzahl vom Benutzer ausgelöster Ereignisse reagieren. Beispiel mit GUI-Element in Formular:

```
<html>
<head>
<script>
    function check() { ..... }
</script>
</head>

<body>
.....
<input type="button" name="checkButton"
       value="check input" onclick="check()" >
.....

```

check input

Auch hier beliebiger Skript-Code!

- Einschlägige Sprachen für klientenseitig ausgeführten eingebetteten Skript-Code *in HTML*:
 - **JavaScript** mit Varianten
 - ECMAScript
 - Jscript (Microsoft)
 - JScript.NET (Microsoft)
 - **VBScript**
 - **Tcl**
 - **Flash**
- **JavaScript** wird von fast allen Browsern unterstützt!
- **Flash** ist attraktiv für Graphik/Animation

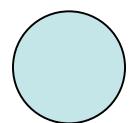
10.5.2 JavaScript - Teil 1

JavaScript ist *keine* Java-Variante!

- Mehrparadigmensprache:
 - *objektbasiert* mit Prototypen statt Klassen
 - *funktional* (beeinflusst durch *Scheme, Self, Smalltalk, ...*)
- ausgeführt durch einen Interpretierer im Browser
- kann umgehen mit Elementen des Dokuments und des umgebenden Browser-Fensters, ferner mit Ereignissen

Meilensteine und Implementierungen:

- JavaScript (1995): Netscape/Sun
- ECMAScript: ECMA (*European Computer Manufacturers Association*)
- JScript: Microsoft
- Seit Version 1.5: Orientierung an **DOM** (*Document Object Model*): standardisierte Repräsentation des Dokumentinhalts



Ein erstes Beispiel:

```
<html><head>
<script>
function action()  {
    document.write("<br><font size=10 color='red'>");
    document.write("<b>autsch!</b> </font> ")
}
</script>
</head>
<body>
<br>Dies ist
<script> document.write("ein Test:<br><br>")
</script>
<input type="button" value="drück mich!" onclick="action()" " >
<br><br>Nein,
<a href="javascript: alert('aua!') ">mich!
</body>
</html>
```

Hilfreich: Fehlermeldungen auf Fehlerkonsole des Browsers

Wichtige Eigenschaften:

- *keine statische Typisierung*
- *Mehrparadigmen-Sprache:*
 - *objektähnliche Elemente (keine Klassen)*
 - *„Vererbung“: Erweiterung eines Prototyps*
 - *funktionale Elemente*
 - *„Funktionen“ sind selbst Objekte*
- *DOM: HTML-Elemente als Objekte*

Objekte haben

- Eigenschaften (*properties*)
- Methoden (*methods*)

und können *ad-hoc* (!) um solche erweitert werden.

Funktionen können als Konstruktoren eingesetzt werden (vgl. Beta):

```
function Complex(r,i) { this.re = r; this.im = i; }
var c = new Complex(3.14, 1.0);
aber hier auch einfach var c = { re: 3.14, im: 1.0 }
```

Kein Unterschied zwischen **Feld**, **Verbund**, **Objekt** (vgl. PHP):

```
c.re == c["re"]
```

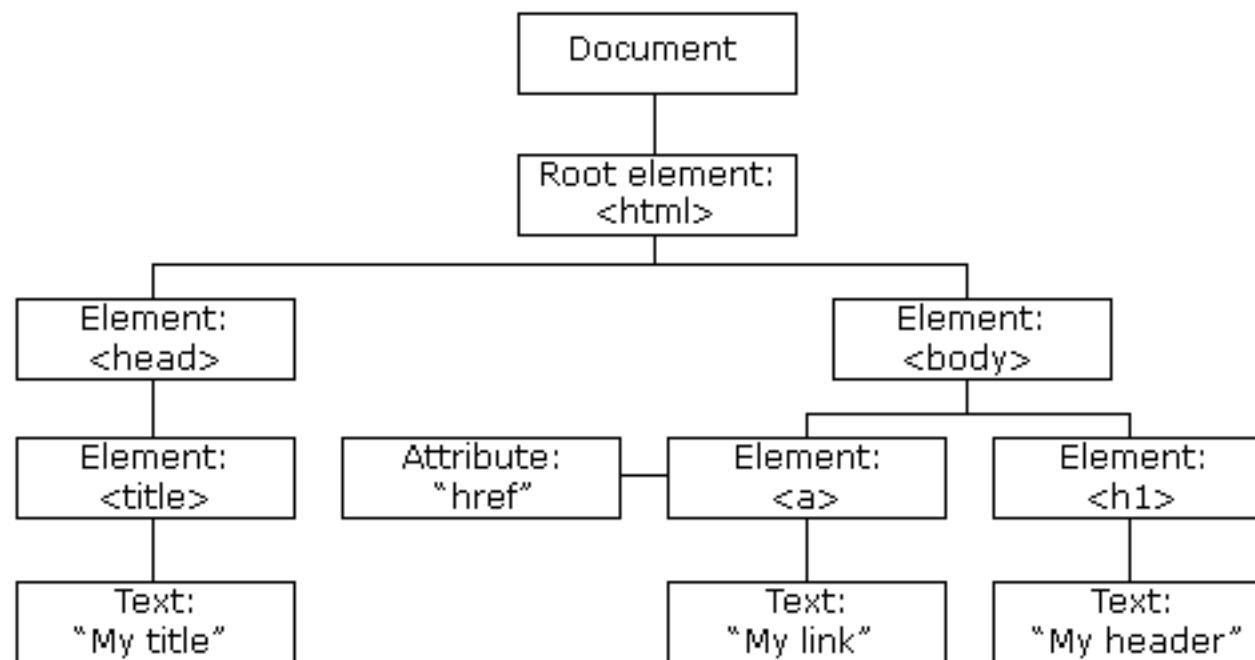
10.5.3 Document Object Model (DOM)

- JavaScript „sieht“ den HTML-Text als Baum von Objekten und kann diesen Baum manipulieren!
- Knoten in diesem Baum sind HTML-Elemente, HTML-Elementattribute und andere.
- Änderungen am Objektbaum schlagen sich sofort in entsprechenden Änderungen der Ansicht nieder!
(Allerdings ist nicht alles erlaubt!)

Beispiel (aus <http://www.w3schools.com/html/dom/default.asp>)

```

<html>
  <head>
    <title>My title</title></head>
  <body>
    <a href="other.html">My link</a>
    <h1>My header</h1></body></html>
  
```



Hier 4 Knotentypen:

- Document
- Element
- Attribute
- Text

„Typen“ von Knoten im Objektbaum:

Node - Obertyp aller Knoten

Document - Wurzel des Baums: `document`

Element - HTML-Element

Attr - Attribut eines HTML-Elements

Weitere allgemeine „Typen“:

NodeList - Liste der Kindknoten eines Knotens

NamedNodeMap - die Attribute eines HTML-Elements

Ferner „Typen“ für etliche spezielle HTML-Elemente.

Navigieren im Objektbaum:

Eigenschaften aller Knoten:

`childNodes` - Feld der Kindknoten

`firstChild` - erster Kindknoten

`nextSibling` - nächster Geschwisterknoten

`nodeName` - Tag-Name, z.B. "A" für `<a ...> ... `
(bei Attributknoten der Attributname)

.....

Navigieren im Objektbaum:

Eigenschaften von `document`:

- `doctype` - Document Type Object mit Eigenschaft `name`
mit Werten wie `"html"`, `"xml"`, `"plain"`
- `images` - Feld mit allen `img`-Elementen
-

Methoden von `document`:

- `getElementById(string)` - liefert das Element mit dem Attribut `ID=string` oder `null` (wenn nicht vorhanden) oder `undefined` (wenn mehrdeutig)
- `getElementsByName(string)` - liefert ein Feld mit allen Elementen mit Attribut `NAME=string`
-

Neue Knoten erzeugen:

Methoden von `document`:

`createElement(string)` - erzeugt einen Knoten,
der einem Element gemäß `string` entspricht,
z.B. `document.createElement("BUTTON")`

`createAttribute(string)` - erzeugt einen Knoten,
der einem Attribut `string` entspricht

`createTextNode(string)` - erzeugt einen Knoten
mit dem angegebenen Text

.....

Knoten in Baum einhängen:

Methoden aller Knoten:

`appendChild(node)` hängt neuen Kindknoten hinten an
`insertBefore(node1, node2)` fügt bei den Kindknoten
node1 vor node2 ein
`removeChild(node)` entfernt angegebenen Kindknoten
.....

Methoden von Element:

`setAttributeNode(attrnode)` Attributknoten an den
Elementknoten anhängen
.....

Zugriff auf Knoteninhalte - lesen und schreiben:

Eigenschaften von `Node`:

`nodeValue` - Wert des Knotens (abhängig vom Typ)

Eigenschaften von `Attribute`:

`name` - Name des Attributs

`value` - Wert des Attributs

Eigenschaften von `Element`:

`tagName` - genau das, z.B. "P" für `<p>...</p>`

Eigenschaften von `HTMLElement`:

`id` - Wert des `ID`-Attributs

`title` - Wert des `TITLE`-Attributs

`innerHTML` - Wert des eingeschlossenen HTML-Textes

Zugriff auf Knoteninhalte - lesen und schreiben:

Methoden von `Element`:

`getAttribute("attr")` liefert den Wert des

Attributs mit dem Namen `ATTR`

`setAttribute("attr", "value")` setzt den Wert

des Attributs mit dem Namen `ATTR` auf `value`

.....

(keine Methoden von `Attribute`)

Zugriff auf Knoteninhalte - lesen und schreiben:

Eigenschaften von diversen speziellen HTML-Knoten, z.B.

Button: type, name, value, ...

Form: name, action, method, elements, ...



Feld aller Elemente des Formulars

Image: name, src, height, width, ...

.....

Events als Attribute von HTML-Elementen:

Name = Event-Name

Wert = *JavaScript-Code*, der beim Eintreffen des Ereignisses ausgeführt wird. **this** ist das dem Element entsprechende DOM-Objekt!

Typische Events:

onclick

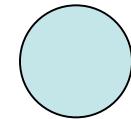
onmousedown onmouseup

onmouseover onmouseout

onsubmit

.....

Jedes Element hat seine eigenen - jeweils sinnvollen - Events.



Beispiel mit einfachem Textelement:

```
<html><body>
<p id="test"
    onclick=" firstChild.nodeValue = 'pong' ">
ping
<br>
pong
</p>
</body></html>
```

Beim Klicken irgendwo im Absatz `test`
wird `ping` durch `pong` ersetzt.

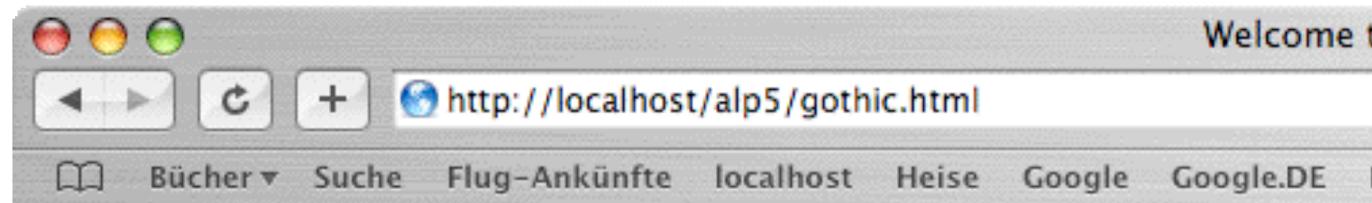
Lokale Überprüfung von Formulareingaben

Zur Erinnerung: `gothic.html` (10.3.3), `gothic.php` (10.3.4)

Zur Entscheidung über eine mögliche Mitgliedschaft im
Gothic Club bedarf es gar keiner Rückfrage beim Server!
Sie kann klientenseitig mit JavaScript vorgenommen werden:

1. Eingabe syntaktisch korrekt?
2. Entscheidung über *eligible*.

Achtung: Trotzdem ist es ratsam, die Prüfung immer auch
beim Server vorzunehmen; denn der Klient kann
ja den Browser umgehen.



Welcome to the Gothic Club!

Check your eligibility for membership:

First name: Last name:

Age:

email:

Sex: male female

```
<FORM action="http://localhost/alp5/php/gothic.php" method="GET">
    First name: <INPUT type="text" name="firstname" size=10>
    Last name: <INPUT type="text" name="lastname"><br>
    Age: <INPUT type="text" name="age" size=2><br>
    email: <INPUT type="text" name="email"><br>
    Sex: <INPUT type="radio" name="sex" value="Male"> male
          <INPUT type="radio" name="sex" value="Female"> female
    <BR><BR>
    <INPUT type="submit"> <INPUT type="reset">
</FORM>
```

ersetzen durch:

partielle Eingabeüberprüfung

```
<FORM onSubmit=">
    var years = parseInt(age.value);
    var error = isNaN(years) || years<=0 || years>150;
    if (error) errorAlert(age.value)
    else check(years,elements[4].checked,firstname.value);
    return false " >

First name: <INPUT type="text" name="firstname" size=10>
Last name: <INPUT type="text" name="lastname"><br>
Age: <INPUT type="text" name="age" size=3><br>
email: <INPUT type="text" name="email"><br>
Sex: <INPUT type="radio" name="sex" value="Male"> male
      <INPUT type="radio" name="sex" value="Female"> female
<BR><BR>
<INPUT type="submit"> <INPUT type="reset">
</FORM>
```

- Das `INPUT`-Element mit Attribut `name="age"` wird im DOM durch das Kind `age` des Formularobjekts repräsentiert und der Wert von dessen `value`-Attribut durch `age.value` (entsprechend für `firstname`).
- `elements[4]` repräsentiert den Schalter `Male`. Das Attribut `checked` prüft, ob er eingeschaltet ist.
- `onSubmit`: Der hier angegebene JavaScript-Code bildet den Rumpf einer anonymen Boole'schen Methode des Formularobjekts. Er wird aktiviert, wenn der `submit`-Knopf gedrückt wird.
- Der gelieferte Wert entscheidet, ob das `submit` tatsächlich ausgeführt wird oder nicht.

```
<SCRIPT>
function errorAlert(age) {
    document.bgColor = "red";
    alert("incorrect age " + val);
    document.bgColor = "white"; }
</SCRIPT>

<SCRIPT>
function check(age, male, name) {
    var eligible = male? age>70 : age>75;
    document.open();
    if(eligible)
        document.write('<b>Ok!</b>')
    else document.write(
        'Sorry, ' + name.split(' ')[0] +
        ', you are not eligible.');
    document.close()          }
</SCRIPT>
```

(Dies z.B. im **HEAD**-Element unterbringen.)

JavaScript Code statt Hyperlink

Script-Aktivierung auch durch Klicken auf Verweis-Element `a`:

```
<a href="javascript: ... (JavaScript code)...">  
... HTML: Text, Bilder, ...  
</a>
```

Beim Anklicken wird der Code ausgeführt.

Aber Achtung: Wenn der letzte berechnete Wert (Ausdruck oder Anweisung) nicht `undefined` ist, ersetzt er - als HTML interpretiert - das aktuelle Dokument.

Beispiel *Bild ersetzen*:

```
<html><head><title>Test DOM</title></head>
<body><br><br>
<p align="center">
<a href="javascript: image.src = 'tree.gif'; undefined ">
    </a>
    <br>
    Open door!
</p>
</body></html>
```

tuer.html

Ohne `undefined` wird der ganze Fensterinhalt
durch den Text „tree.gif“ ersetzt!

Beispiel DOM-Baum verändern: Textknoten entfernen

```
<html><head><title>TestX DOM</title></head>
<body><br><br>
<script> function swap() {
            image.src = 'tree.gif';
            p.removeChild(p.childNodes[4]) }
</script><br><br>
<p id="p" align="center">
    <a href="javascript: swap()">
        </a>
    <br>
    Open door!
</p>
</body></html>
```

tuerX.html

10.5.4 JavaScript - Teil 2

Drei JavaScript-Prinzipien:

- ① **Verwendung:** Reaktion auf Ereignisse, die durch Benutzer-Interaktion ausgelöst werden.
- ② **Bevorzugter Stil:** weitgehende Trennung von HTML und JavaScript: Code in *functions* konzentrieren und z.B. im Dokumentkopf unterbringen - vorzugsweise in getrennten <SCRIPT>-Elementen.
- ③ **Nachgeladener Code** und Trennung von HTML, CSS, JS

Nachgeladener Code

Vollkommene Trennung der *functions* vom Dokument:

```
<html><head><title>Test DOM</title></head>
<body>
<script src="treescrypt.js"></script>
<br><br>
<p id="p" align="center">
<a href="javascript: swap() ">
     </a>
<br>
    Open door!
</p>
</body></html>
```

```
function swap() {
    image.src = 'tree.gif';
    p.removeChild(p.childNodes[4]) }
```

Trennung von HTML, CSS, JS :

Inhalt

(contents)

HTML

Verhalten

(behaviour)

JavaScript

Darstellung

(presentation)

CSS

```
<head>                                myJS.js           myCSS.css
<script src="myJS.js">
<link rel="stylesheet"
      type="text/css"
      href="myCSS.css">
..... (Inhalt!) .....
```

Sicherheitsmechanismen

Mobiler Code kann immer auch **Trojanischer Code** sein!

Beschränkung der Zugriffsmöglichkeiten von JavaScript Code:

- Code läuft in eingeschränkter Umgebung (*sandbox*):
Zugriff nur auf *Browser-* und *DOM-Objekte*
- Weitere Einschränkung (*same-origin policy*):
Zugriff nur auf solche Ressourcen, die zum gleichen
Ursprungsort (protocol, host, port) wie der Code gehören

Weiterreichende Rechtevergabe

jenseits der genannten Einschränkungen ähnlich wie beim Sicherheitssystem von Java (8.5), aber wenig genutzt:

- **Signierter Code**
- **Codebasis**

<http://www.mozilla.org/projects/security/components/jssec.html>

Dennoch:

Schwachstellen im Web - insbesondere ausnutzbar durch Skript-Code - sind nach wie vor die beliebtesten Einfallstore bei Angriffen auf Informatik-Systeme.

Kleine Fallstudie

Beim Formular des *Gothic Club* soll Anna ihren Namen eingeben.
Was passiert, wenn sie stattdessen folgendes eingibt?

```
<script>alert( "bang! " )</script>
```

Wenn Anna zu jung für den Club ist, erhält sie dann die Antwort „Sorry, `<script>alert("bang!")</script>`,“ ?

- ① Nein, wenn die Eingabe-Überprüfung diese Eingabe ablehnt.
- ② „Ja, wenn dieser Text *nicht überprüft wird*.“
- ③ Nein, doch nicht, weil `document.write` HTML-Text als solchen interpretiert, d.h. das Skript erkennt und `alert` ausführt: die `alert`-Meldung „bang!“ erscheint.

Damit schadet Anna nur sich selbst. Es liegt also noch keine Sicherheitsverletzung vor.

- ④ Was passiert, wenn *nicht lokal* mit JavaScript geprüft wird, sondern - wie in der ursprünglichen Version (10.3.4) - beim Server mit PHP? Der Name wird URL-codiert
%3Cscript%3Ealert%28%22bang%21%22%29%3C%2Fscript%3E im *query string* an den Server geschickt und dort decodiert. Wenn der Name dann *nicht überprüft wird*, wird das Skript-Element in die Antwortseite eingebaut - aber mit \" statt " -
Sorry, <script>alert(\"bang!\")</script>, ... und daher wegen Funktionsunfähigkeit ignoriert.

Kann Anna durch „ähnlich verrückte“ Eingaben beim Server oder anderen Klienten Schaden anrichten?

Cross-Site Scripting (XSS) :

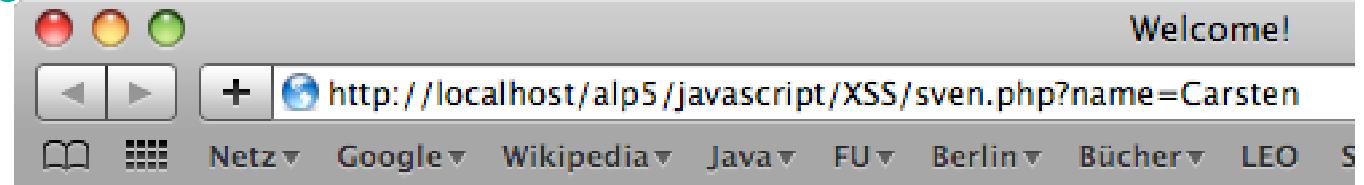
Angreiferin Anna kann eine Seite mit **Schwachstellen** von Sven dazu benutzen, einem Dritten Carsten, der diese Seite besucht, bösartigen Skriptcode unterzuschieben.

Die Mehrzahl aller Sicherheitsverletzungen im Netz ist auf XSS (oder ähnliche *code-injection*-Techniken) zurückzuführen!

Ältere Server- und Browser-Software ist wesentlich anfälliger für XSS als neuere Versionen.

Svens schwache Webseite <http://cool.tools.com/sven.php>:

```
<html><head><title>Welcome!</title></head><body><br>
Hello
<?      $name = $_GET[ ' name' ];
        $name = str_replace( '\\\\', ' ', $name);    sehr riskant!
        if($name != ' ') {
            echo "<script>document.write( '$name' );";
            echo "</script>";    }    ?>
<br><br>You can choose from:<br>....<br>....</body></html>
```



Hello Carsten!

You can choose from:

.....

Angreiferin *Anna* lockt *Carsten* auf ihre Webseite *anna.html*:

```
<html><title>Welcome!</title><br>
Hello friend!<br> <br>
Get cool tool from
<a href="sven.php?name=' );alert('Sorry, expired Dec. 2010!">
    Sven!
</a>
</html>
```

anna.html

Ärgerlich für *Carsten* und geschäftsschädigend für *Sven!*
„Geschieht ihm recht!“)

Vermischtes:

- JavaScript-Code wird häufig von Entwicklungsumgebungen automatisch erzeugt.
- JavaScript kann als Web-unabhängige Skriptsprache verwendet werden. Beliebte Interpretierer sind *Rhino* und *SpiderMonkey*
- Server-Side JavaScript (SSJS):
`<server> write("
hello"); </server>`
- Java-Programm kann JavaScript-Code ausführen:
`package javax.script, class ScriptEngine, eval(...),`
- JavaScript-Code kann Java-Objekte erzeugen, benutzen:
`„LiveConnect“ (Objekt Packages mit Objekt Packages.java.lang)`

10.5.5 Applets etc.

- Applets sind (kleinere) Java-Programme, die in einem Java-fähigen Web Browser gestartet werden können.
- Ein HTML-Element `<object>` oder `<applet>` oder `<embed>` in einem Dokument enthält den Namen einer Datei, aus der das Applet *nachgeladen* werden kann, z.B.
`<applet code="xyz.class"`
- Applets waren einmal ein großer Hype, sind jetzt wegen leistungsfähiger JavaScript-Implementierungen weniger populär - aber immer noch für viele Zwecke geeignet.

Dokument-Struktur mit Applets:

```
<html><head> .... </head>
<body>
.....
<object .... >
    <param ... >
    <param ... >
    .....
</object>
.....
<object .... >
    .....
</object>
.....
</body></html>
```

belegt einen Bereich im Browser-Fenster

belegt einen Bereich im Browser-Fenster
weiter unten

Ein Applet ist eine Unterklasse von `java.applet.Applet` oder `javax.swing.JApplet`.

Nach dem Laden eines Applet erzeugt der Browser ein entsprechendes Objekt und „steuert“ es über bestimmte Methoden, die in `Applet` vordefiniert sind:

void init() Initialisierung, Übernahme von Parametern aus `<param>` (ähnlich Konstruktor).
Vordefinition: Leeroperation.

void destroy() Finalisierung, Freigabe von Ressourcen.
Vordefinition: Leeroperation.

Beachte: Die Lebensdauer eines Applet ist i.d.R. länger als die des einbettenden Dokuments. Der Browser entscheidet, wann er es löscht (mit vorangehender Finalisierung).

public void start()

- wird vom Browser nach `init` aufgerufen
- und immer dann, wenn der Benutzer zur Seite zurückkehrt, die das Applet enthält
- wird insbesondere für das Starten von Animation eingesetzt

public void stop()

- wird vom Browser vor `destroy` aufgerufen
- und immer dann, wenn der Benutzer die Seite verlässt, die das Applet enthält
- wird insbesondere für das Stoppen von Animation eingesetzt

```
public void paint(Graphics g)
```

- stammt aus der Oberklasse `java.awt.Container`
- wird zur Darstellung von Graphik verwendet
- das `Graphics`-Objekt stellt der Browser zur Verfügung
- wird nach jedem `start` aufgerufen
- ... und immer dann, wenn durch manuelle Änderungen am Fenster (Größenveränderung, Ikonisierung) der Inhalt neu angezeigt werden muss
- ... und immer dann, wenn das Applet `repaint()` aufruft

Diverse weitere Methoden, u.a. vom Applet aufzurufende (!)
- für Parameterübernahme, Kontextinformation etc.

- Die Klasse `Applet` ist eine mittelbare Unterklasse der Klasse `Component` - und damit ein **GUI-Fenster** !
- Somit stehen alle Methoden der Klasse `Component` zur Bearbeitung von **Ereignissen** zur Verfügung !
- Beispiele für Ereignisklassen: `KeyEvent`, `MouseEvent`, ...
(Unterklassen von `AWTEvent`)

```
import java.awt.*;
import java.awt.event.*;          // simple GUI applet
import java.applet.*;
public class Interaction extends Applet {
    private TextField text = new TextField();
    private Button button = new Button("show");
public void init() {
    this.setLayout(new GridLayout(2,1));
    this.add(text);
    this.add(button);
    Rectangle r = this.getBounds(); // from <object..>!
    this.setFont(new Font(
        "Helvetica", Font.PLAIN, r.height/5));
    ActionListener action = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            showStatus(text.getText()); } };
    button.addActionListener(action); } // end init
} // end class
```

... und das einbettende Dokument `inter.html`:

```
<HTML><TITLE>Applet interaction</TITLE>
<br>
<h2>Here comes an Applet</h2>
<font size="5">Please enter text and press "copy". <br>
          Text will be copied into status line. <br>
</font><br>
<object codebase=". "
           classid ="java:Interaction.class"
           codetype="application/java-vm"
           width="200"  height="100"  alt="broken!">
  <p> A GUI Applet should be displayed here,
      but your browser does not support applets.
  </p>
</object>
</body></html>
```

(← allgemein: eine URL)

*Beachte: Test auch mit `$ appletviewer inter.html`
oder Java-main mit `init()`- und `start()`-Aufrufen.*

Beliebt ist die Implementierung von **Animationen** mit Applets:

- **public class MyMovie extends Applet implements Runnable**
- **public void run() { } bereitstellen**
- Thread erzeugen mit **new Thread(this)**
- Es empfiehlt sich, **start/stop** geeignet zu implementieren.

Sicherheit von Applets:

Welchen Schaden können **Trojanische Applets** anrichten?

Applet kann auf Ressourcen der Umgebung nur sehr beschränkt zugreifen, sofern es nicht **signiert** ist und weiterreichende Rechte - z.B. für Dateizugriff - erhält (siehe Quellen).

Ein Applet

- kann nur mit dem System, von dem es geladen wurde, TCP-Verbindungen herstellen oder akzeptieren,
- kann nur wenige *system properties* lesen
- (und einige weitere harmlose Dinge).

Beispiel:

Gewinnen Sie hier eine Reise nach Hawaii

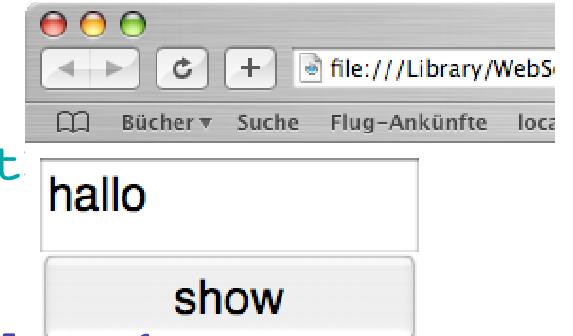
(Applet versucht, in Datei zu schreiben. Effekt:)

```
java.security.AccessControlException:  
access denied ( java.io.FilePermission  
/Users/lohr/tmp/text write)
```

LiveConnect: Applets mit JavaScript und DOM

① JavaScript kann auf **public**-Elemente von Applets zugreifen!

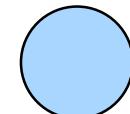
```
<html><head><title>JS to Applet</title></head>
<body>
<applet id="applet" code = "Interaction.class"
        width="200" height="100" >
</applet>
<script> applet.write("hallo"); </script>
</body></html>
```



```
public class Interaction extends Applet {
    private TextField text = new TextField();
    private Button button = new Button("show");
    public void init() { .....(S. 50)..... }
    public void write(String s) { text.setText(s); }
}
```

Beispiel Stoppen und Neustarten einer Animation:

```
<html><head><title>Stoppable drop</title></head>
<body>
<applet id="applet" code ="Drop.class"
        width="300" height="300" >
</applet>
<input type="button" value="Start"
       onClick="applet.start()">
<input type="button" value="Stop"
       onClick="applet.stop()">
</body>
</html>
```



Wen es interessiert:

```
import java.applet.*;
import java.awt.*;
public class Drop extends Applet implements Runnable {
    private volatile boolean stop;
public void start() { stop = false; new Thread(this).start(); }
public void stop() { stop = true; }
public void run() {
    Graphics g = this.getGraphics();
    Rectangle re = this.getBounds();          // Applet-Bereich
    int r=0,                                // aktueller Radius
        d,                                    // aktueller Durchmesser
        width=re.width/2,                     // halbe Applet-Breite
        height=re.height/2;                  // halbe Applet-Höhe
    while(true) {
        if(stop) return;
        if(r>Math.max(width,height)) r = 0;
        d = 2*r;
        g.clearRect(0,0,re.width,re.height);   // Kreis löschen
        g.drawOval(width-r,height-r,d,d);     // neuer Kreis
        g.drawOval(width-(r-1),height-(r-1),d-2,d-2);
        r++;                                 // 2 Kreise für "Linienbreite 2"
        try { Thread.sleep(50); }
        catch (InterruptedException e) { }    } // end loop
    } // end run()
} // end class Drop
```

- ② `<applet mayscript ...>` kann *JavaScript Code* ausführen, die *JavaScript-Funktionen* im Dokument aufrufen und auf die *DOM-Objekte* zugreifen! Diese haben den Typ `netscape.javascript.JSObject`.
(`$ javac -cp .:$JAVAHOME/lib/plugin.jar ...` !)

```
.....  
window = JSObject.getWindow(this);  
document = (JSObject)window.getMember("document");  
.....  
window.eval(" ..... JavaScript code ..... ");  
window.call(". JavaScript function name .", argArray);  
.....  
JSObject node = (JSObject)  
    document.call("../ document method name ..", argArray);  
.....
```

(Alternative: **Common DOM API** - siehe Quellen)

Java „Rich Internet Applications“ (RIA)

subsumieren Applets und Java Web Start,
verwenden das Java Network Launching Protocol (JNLP),
ermöglichen flexible Steuerung der Sicherheit.

Java Web Start:

Java-Programme werden von einem Web Server
heruntergeladen und gestartet.

Dateien beim Server: `prog.html`, `prog.jar`, `prog.jnlp`
(siehe Quellen)

Andere Medientypen mit <object>

Viele andere **MIME Types** (Format type/subtype) neben application/java-vm sind möglich, etwa

audio/midi
application/x-shockwave-flash
image/vnd.adobe.photoshop
text/plain und viele andere

Typisches Beispiel:

```
<object data="http://ingeb.org/Lieder/freudesc.mid"  
       type="audio/midi"  
       width="100" height="50"  
       ....  
</object>
```



```
<html><body><br>          Beispiel mit 3 Objekten
  <h4>An die Freude</h4>
  <object data="schiller.txt"
          type="text/plain"
          width="300" height="100" border="1">
</object>
  <object data="http://ingeb.org/Lieder/freudesc.mid"
          type="audio/midi"
          width="100" height="50">
</object><br>          macht Ausschnitt aus Radio-Bild sichtbar
  <object data="http://www.youtube.com/v/NKusg6Jyc9Y&rel=0"
          type="application/x-shockwave-flash"
          width="700" height="520">
</object>
</body></html>
```

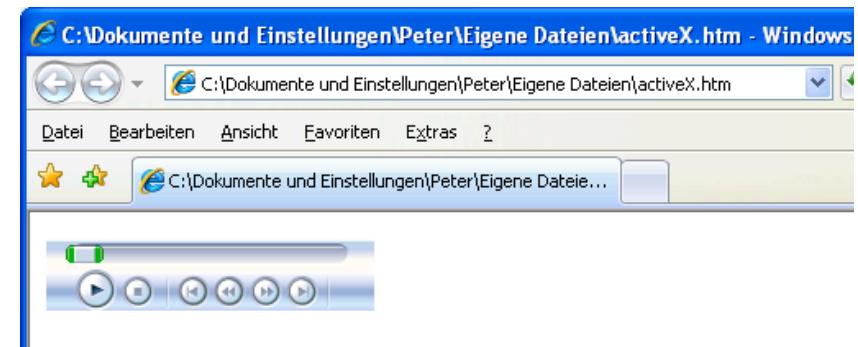
`type` ist entbehrlich, wenn die Dateierweiterung
hinreichend Aufschluss gibt (so hier).

Microsoft ActiveX Controls

- können ähnlich wie Applets eingesetzt werden:
- ursprünglich nur *Maschinencode* für Intel,
- mit .NET auch für die virtuelle CLR-Maschine,
- heruntergeladen vom MS *Internet Information Server*,
- durch den MS *Internet Explorer*.

Beispiel Media Player:

```
<object codebase="" (URL - hier entbehrlich, weil Media Player lokales Plugin) "
    classid="CLSID:05589FA1-C356-11CE-BF01-00AA0055595A"
    width="200" height="45" >
    <param name="filename"
        value="http://ingeb.org/Lieder/freudesc.mid">
</object>
```



10.6 AJAX

(„Asynchronous JavaScript and XML“)

Resümee soweit:

Web-Anwendung (*Web application*) besteht aus

- Anwendungslogik beim Server und
- Anwendungslogik und GUI beim Klienten,
mobil, vom Server dynamisch heruntergeladen
- ! Auf beiden Seiten sind potentiell mehrere,
verschiedene Programmiersprachen beteiligt !
- Die GUI-Möglichkeiten sind durch HTML und
durch die Applet/ActiveX-GUI-Elemente gegeben.
- Dürftige Interaktionsmöglichkeiten zwischen
Backend und *Frontend*: *submit* bewirkt HTTP-
Anfrage und dann Aufbau eines neuen Dokuments.

AJAX :

- JavaScript Code im Dokument kann *selbst* eine **HTTP-Anfrage** starten, deren Antwort das aktuelle Dokument nicht ersetzt, sondern nur modifiziert (*aber same-origin policy!*).

- Anfrage vorbereiten: *XMLHTTP-Objekt erzeugen*, dessen Konstruktorfunktion von der Umgebung abhängt:

```
if(window.ActiveXObject)          // Internet Explorer
    httpRequest = new ActiveXObject("Microsoft.XMLHT
else if(window.XMLHttpRequest)// Firefox, Safari, ...
    httpRequest = new XMLHttpRequest();
```

- Eine *Listener-Funktion f* definieren, die festlegt, was beim Eintreffen der Antwort zu tun ist, und damit:

```
httpRequest.onreadystatechange = f
```

Asynchrone Anfrage starten:

```
http_request.open('GET',  
                  'http://www.xyz/some.file', true);  
http_request.send(null);
```

(asynchron)

auch *query string* statt `null` bei Verwendung von POST

Reaktion auf die Antwort:

`http_request.readyState` von 0 bis 4 !

Bei jeder Zustandsänderung wird die Funktion `f` aufgerufen;
sie muss den aktuellen Zustand prüfen:

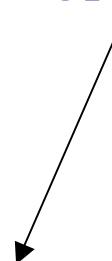
```
if(http_request.readyState == 4) {  
    . . . . . // response received  
} else { . . . . . // still not ready
```

0 (uninitialized)
1 (loading)
2 (loaded)
3 (interactive)
4 (complete)

```
var request, i = 0;
function makeRequest(url) {
    request = new XMLHttpRequest();
    request.onreadystatechange = updateDocument;
    request.open("GET", url, true);
    request.send(null);
}
function updateDocument() {
    if(request.readyState != 4) return false;
    if(request.status == 200) {
        var div = document.createElement("div");
        div.innerHTML = request.responseText;
        var span = document.getElementById("span");
        span.appendChild(div); }
    else alert("failure - illegal request");
}
```

```
<span id="span"
      style="cursor: pointer;
              font-size: 20;
              background-color: aqua"
      onclick=" makeRequest('pingpong.html') " >
    Make a request!
</span>
```

ajax.html



```
<p id="pingpong"
    onclick=" firstChild.nodeValue = 'pong' ">
  ping <br> pong
</p>
```

Ein „universelles“ `getResponse` für verschiedene Formate:

```
function getResponse(request) {  
    switch(request.getResponseHeader("Content-Type")) {  
        case "text/plain":  
        case "text/html": return request.responseText;  
        case "text/xml": return request.responseXML; // liefert DOM!  
        case "application/json":  
            return jsonParse(request.responseText);  
            // http://code.google.com/p/json-sans-eval/  
        case "application/javascript":  
            return eval(request.responseText);  
            // potentiell unsicher (Daten als Code)  
        default: return request.responseText;    }  
}
```

Zusammenfassung

- *Mobiler Code*: der Web Server kann nicht nur Daten liefern, sondern auch Code - vorzugsweise JavaScript - der im Browser ausgeführt wird.
- *Document Object Model (DOM)*: JavaScript kann auf Elemente des Dokuments als vordefinierte Objekte in einem Objektbaum zugreifen.
- *Sicherheit*: Web-Anwendungen sind berühmt-berüchtigt für Sicherheitslücken, von denen viele auf Probleme mit Skriptsprachen zurückzuführen sind.

- *Mobiler Code:* Applets sind „kleine“ Java-Programme, die wie Bilder und andere Objekte nachgeladen werden. Typische Browser sind „java-fähig“, d.h. sie führen das Applet in einer JVM aus.
- *Sicherheit:* Ein Applet darf auf „fast gar keine“ externen Ressourcen zugreifen, außer wenn ihm explizit über eine Richtliniendatei weitergehende Rechte verliehen wurden (insbesondere bei signierten Applets).
- *JavaScript Code kann die öffentlichen Methoden von Applets aufrufen.* Umgekehrt (aber weniger verbreitet) können Applets vom der umgebenden JavaScript-Welt Gebrauch machen (Interpretierer und Code).

- AJAX: Nachladen von HTML-Fragmenten mit HTTP GET oder POST über ein `XMLHttpRequest`-Objekt; auch andere Medientypen als HTML sind möglich.

Quellen

St. Münz: *Web-Seiten professionell erstellen* (3.Aufl.). Addison-Wesley 2008
(und SelfHTML: <http://de.selfhtml.org/javascript>)

D. Flanagan: *JavaScript* (5.ed.). O'Reilly 2006

Mozilla Developer Center: *JavaScript*.
<https://developer.mozilla.org/en/JavaScript>

-

W3C: *Document Object Model (DOM)*
<http://www.w3.org/TR/DOM-Level-3-Core>
<http://www.w3schools.com/jsref/>
<http://www.w3schools.com/htmldom/default.asp>

Mozilla Developer Center: *JavaScript Security*.
<http://www.mozilla.org/projects/security/components/jsssec.html>

- Applets: <http://docs.oracle.com/javase/tutorial/deployment/applet/>
https://en.wikipedia.org/wiki/Java_applet
www.oracle.com/technetwork/java/javase/plugin2-142482.html
- <applet> tag: http://docs.oracle.com/javase/1.4.2/docs/guide/plugin/developer_guide/using_tags.html
- LiveConnect: <http://jdk6.java.net/plugin2/liveconnect/>
- Common DOM API: <http://docs.oracle.com/javase/7/docs/jre/api/plugin/dom/>
http://docs.oracle.com/javase/6/docs/technotes/guides/plugin/developer_guide/java_js.html
- Appletviewer: docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/appletview
- Applet security: docs.oracle.com/javase/tutorial/deployment/applet/security.htm
<http://docs.oracle.com/javase/tutorial/security/index.html>

Java Web Start: <http://www.oracle.com/technetwork/java/javase/javawebstart>
<http://docs.oracle.com/javase/6/docs/technotes/guides/javaws/developersguide/contents.html>
<http://docs.oracle.com/javase/6/docs/technotes/guides/jweb/>

MS ActiveX: <http://www.fpoint.com/support/whitep/fp-ActiveX-toWeb.pdf>

AJAX: <http://www.w3.org/TR/XMLHttpRequest/>
[http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming))