Datenbank-basierte Webserver

Client-seitige Operationen unter Verwendung von JavaScript

Karl Riedling
Institut für Sensor- und Aktuatorsysteme

Client-seitige Operationen mit JavaScript

- Anwendungsbereiche von JavaScript
- Grundlegende Eigenschaften von JavaScript
- Formulare und JavaScript
- Javascript-*Event Handlers*
- AJAX (Asynchronous JavaScript and XML)
- JavaScript-Frameworks
- JavaScript am Client-Rechner

Client-seitige Operationen mit JavaScript

- **Anwendungsbereiche von JavaScript**
- Grundlegende Eigenschaften von JavaScript
- Formulare und JavaScript
- Javascript-*Event Handlers*
- AJAX (Asynchronous JavaScript and XML)
- JavaScript-Frameworks
- JavaScript am Client-Rechner

Anwendungen von JavaScript

- Client-seitige Datenverarbeitung sinnvoll für:
  - Dynamische Veränderung von Seiten-Inhalten (z.B. *Rollovers*)
  - (Lokale) Reaktion auf User-Aktionen
  - **Vorverarbeitung und/oder Prüfung von Formularinhalten**

Anwendungen von JavaScript

- Prüfung von Formularen
  - Pflicht-Formular-Felder ausgefüllt?
  - Sinnvoller Inhalt (z.B. korrektes Format für Datum, numerische Werte, wo benötigt)?

Anwendungen von JavaScript

- Vorverarbeitung von Formular-Inhalten vor dem Abschicken
  - Ersetzen von Zeichen durch Escape-Sequenzen o.ä.
  - Prüfsummen, irgendwelche Berechnungen
- JavaScript-generierte Inhalte können in `<input type="hidden">`-Formularfeldern an den Webserver übergeben werden

Anwendungen von JavaScript

- Voreinstellung und Modifikation von Formular-Inhalten
  - Setzen oder Löschen von *Checkboxes*, *Radio Buttons* usw. in Abhängigkeit von anderen Einstellungen
  - Manipulation des Inhalts einer Auswahlliste in Abhängigkeit von der Einstellung eines anderen Formularfeldes

Anwendungen von JavaScript

- Automatisches Abschicken von Formularen
  - Neuaufbau des Formulars bei Änderungen (Alternative zu lokaler Manipulation der Formulardaten)

Anwendungen von JavaScript

- Automatisches Abschicken von Formularen
  - Zweckmäßig, wenn Datenmenge im Formular (speziell Auswahllisten) begrenzt werden soll
  - Lokale Manipulation einer Auswahlliste erfordert, dass ursprünglich ***alle*** potenziell möglichen Optionen übertragen werden müssen; bei Neuaufbau werden nur die tatsächlich benötigten Daten übertragen!

Anwendungen von JavaScript

- Alternative zum automatischen Abschicken von Formularen: AJAX (Asynchronous JavaScript and XML)
  - AJAX basiert auf in den meisten modernen Browsern vorhandenen Funktionalitäten und JavaScript
  - Vorteile:
    - Geringere Datenmengen zu übertragen
    - „Lückenloser“ Betrieb einer Webseite (ohne die durch das Neuladen auftretenden Pausen

Anwendungen von JavaScript

- Lokale Such-Funktionen
  - Suche nach bestimmten Einträgen in einer umfangreichen Auswahlliste
- „Verriegelung“ eines Formulars gegen vorzeitiges oder mehrfaches Abschicken

Anwendungen von JavaScript

- Client-seitige Datenverarbeitung ***nicht*** sinnvoll bzw. geeignet für:
  - Sicherheits-sensitive Aktivitäten (z.B. Validierung von Passworten)
  - Direkten Zugriff auf die Datenbank

Anwendungen von JavaScript

- JavaScript läuft auf dem Client-Rechner als Funktion des Browsers und ist daher in seinem Verhalten weniger vorhersehbar und testbar als Server-seitige Software
- Im Falle Client-seitiger Daten-Vorverarbeitung Ergebnisse jedenfalls auf Plausibilität prüfen!

Anwendungen von JavaScript

- Viele Features von JavaScript sind nur unter bestimmten Browsern verfügbar – Browser-Versions-Abhängigkeit
- Daher möglichst nur fundamentale JavaScript-Funktionen verwenden!

Client-seitige Operationen mit JavaScript

- Anwendungsbereiche von JavaScript
- **Grundlegende Eigenschaften von JavaScript**
- Formulare und JavaScript
- Javascript-*Event Handlers*
- AJAX (Asynchronous JavaScript and XML)
- JavaScript am Client-Rechner

Eigenschaften von JavaScript

- Allgemeines
- Datentypen
- Variable
- Operatoren
- Konstrukte
- Funktionen
- JavaScript und HTML

Eigenschaften von JavaScript

- **Allgemeines**
- Datentypen
- Variable
- Operatoren
- Konstrukte
- Funktionen
- JavaScript und HTML

Allgemeines

- Objektorientierte Sprache – die meisten JavaScript-internen Funktionen sind *Methoden* von JavaScript-Objekten
- Daten sind grundsätzlich *Properties* eines (geeigneten) JavaScript-Objekts
- *Case sensitive* – Sprachelemente üblicherweise in Kleinschreibung

Allgemeines

- Befehle können sich über mehrere Zeilen erstrecken
- Befehle, die jeweils in separaten Zeilen stehen, müssen nicht unbedingt mit „;“ abgeschlossen werden
- Sicherheitshalber aber JavaScript-Befehle immer mit „;“ abschließen!

Allgemeines

- Kommentare (wie in PHP) mit „/* … */“ (innerhalb des aktiven Programmcodes oder über mehrere Zeilen) oder mit „//“ (Kommentar bis Ende der Zeile)
- Zusätzlicher Einzelzeilen-Kommentar mit „<!--“ – Anwendung: „Verstecken“ von JavaScript-Code vor Browsern, die nicht JavaScript verstehen:

Allgemeines

- Beispiel für JavaScript-Definition:

```
<script language="JavaScript">
<!--  Hier beginnt ein HTML-Kommentar
// Die obige Zeile ist ein JavaScript-
// Einzelzeilen-Kommentar
// Beliebiger JavaScript-Code
// Die nächste Zeile ist das Ende des
// HTML-Kommentars und ein JavaScript-
// Kommentar
// -->
</script>
```


Allgemeines

- Namen in JavaScript:
  - Erstes Zeichen: ASCII-Buchstabe (GROSS oder klein), „_“ oder „$“ („$“ ab JavaScript 1.1); **keine Ziffer!**
  - Weitere Zeichen: Alphanumerische ASCII-Zeichen, „_“ oder „$“
  - Namen dürfen nicht mit reservierten Schlüsselworten (➔ JavaScript-Dokumentation) kollidieren!

Eigenschaften von JavaScript

- Allgemeines
- **Datentypen**
- Variable
- Operatoren
- Konstrukte
- Funktionen
- JavaScript und HTML

Datentypen

- Numerische Werte und Konstanten:
  - Alle Werte, auch ganzzahlige, werden intern als Gleitkommawerte dargestellt (exakte Repräsentation für `-2^53 … +2^53`)
  - Gleitkomma-Wertebereich ca. `±1.8e308` (64-Bit-`double`-Format)
  - Gleitkomma-Darstellung als Dezimalzahl (`3.14`) oder in *Scientific Notation* (`1.602e-19`)
  - Oktale (z.B. `0377`) und hexadezimale Konstanten (z.B. `0xff`)

Datentypen

- *Strings*:
  - Entweder in einfachen („'“) oder doppelten („"“) Anführungszeichen
  - Strings in doppelten Anführungszeichen dürfen einfache Anführungszeichen enthalten und umgekehrt
  - Bei modernen Browsern (nicht bei Netscape 4.x!) unterstützt JavaScript Unicode
  - **Vorsicht**: Keine HTML-*Character Entities* (z.B. „`&auml;`“ für „ä“) in JavaScript-Ausgabetexten verwenden!

Datentypen

- JavaScript-Escape-Sequenzen:

`\b`	*Backspace*	`\f`	*Form feed*
`\n`	*Line feed*	`\r`	*Carriage return*
`\t`	Tabulator	`\'`	Einf. Anführungsz.
`\"`	Dopp. Anf.zeichen	`\\`	*Backslash*
`\`*nnn*	Oktalcode	`\x`*nn*	Hexadezimalcode
`\u`*nnnn*	Unicode (mit 4 hexadezimalen Stellen)		

Datentypen

- Boole'sche Werte:
  - Typisch: Ergebnis von Vergleichsoperationen
  - Zwei Werte, `true` und `false` (repräsentiert als `1` und `0`)
- Benutzerdefinierte Funktionen:
  - Funktionen sind in JavaScript *Datentypen*!

Datentypen

- Eingebaute Funktionen:
  - In der Regel als Methoden eines Objekts implementiert, z.B.:

```
sinx = Math.sin(x);

str = "abcdefgh"; // Objekt der Type "String"
str1 = str.toUpperCase();
    // ergibt "ABCDEFGH"
```

Datentypen

- Objekte:
  - Darstellung der Inhalte von Dokumenten, Fenstern und Formularen als Objekte: `document.myform.button` (Button-Element in Formular `myform` auf der aktuellen Web-Seite)
  - Alternativ interpretierbar als assoziatives Array: `document.myform["button"]`
  - Erstellt durch Aufruf einer Konstruktor-Funktion: `var jetzt = new Date ();`

Datentypen

- Felder (Arrays):
  - Elemente indiziert durch nichtnegativen Indexwert: `document.images[i].width`
  - Feldindizes beginnen immer bei `0`!
  - Elemente eines Feldes können beliebige, auch unterschiedliche, Typen haben

Datentypen

- Null:
  - Schlüsselwort „`null`“ steht für „kein Wert“
  - Oft verwendet im Zusammenhang mit Objekten („kein Objekt“)
  - Meist konvertiert in den Wert „`0`“, aber nicht identisch mit „`0`“!

Datentypen

- *Undefined*:
  - Verwendet in den folgenden Fällen:
    - Nicht existierende Variable wird verwendet
    - Variable wurde zwar deklariert, aber nicht mit einem Wert initialisiert
    - Gewählte Eigenschaft eines Objekts existiert nicht
  - Es gibt ab JavaScript 1.3 eine Property „`undefined`“ (ebenso wie die zuvor nicht definierten Properties „`infinity`“ und „`NaN`“ („*Not a Number*“)), mit der auf „*Undefined*“ getestet werden kann.

Datentypen

- Weitere Datentypen (➔ JavaScript-Dokumentation):
  - *Date*-Objekt
  - Reguläre Ausdrücke

Eigenschaften von JavaScript

- Allgemeines
- Datentypen
- **Variable**
- Operatoren
- Konstrukte
- Funktionen
- JavaScript und HTML

Variable

- Keine fixen Variablen-Typen – die gleiche Variable kann unterschiedliche Datentypen beinhalten
- Bei Bedarf automatische Konversion (z.B. bei Ausdrücken mit Variablen mit unterschiedlichen Datentypen)

Variable

- Variable sollten immer mit dem Schlüsselwort „`var`“ deklariert werden:

```
var i;
var j, k, l;
var x = 123, msg = "hello";
```

- Mehrfache Deklarationen der selben Variablen sind zulässig

Variable

- Variable, die innerhalb einer Funktion mit „`var`“ deklariert wurden, sind nur innerhalb der Funktion gültig (*local scope* – lokale Variable)
- Variable, die verwendet werden, ohne vorher mit „`var`“ deklariert worden zu sein, werden als globale Variable betrachtet – Gefahr des versehentlichen Änderns globaler Daten!

Variable

- *Lesen* einer nicht deklarierten Variablen bewirkt einen Laufzeitfehler (Schutz vor Tippfehlern!)
- *Wertzuweisung* an eine nicht deklarierte Variable deklariert sie implizit
- Deklarierte, aber nicht initialisierte Variable sind `undefined`

Variable

- Fundamentale Variable (z.B. numerische Daten) werden bei Zuweisungen *kopiert*, komplexe Datenkonstrukte (z.B. Felder, Objekte) werden *als Referenzen übergeben*

Variable

- Alle Variablen und Funktionen sind Datenelemente (*properties*) bzw. Funktionen (*methods*) eines Objekts:
  - Globale Variable: *global object*, z.B. `window`
  - Lokale Variable: *call object*

Eigenschaften von JavaScript

- Allgemeines
- Datentypen
- Variable
- **Operatoren**
- Konstrukte
- Funktionen
- JavaScript und HTML

Operatoren

- Arithmetische Operatoren:
  - „+“, „-“, „*“, „/“: Grundrechnungsarten (*immer* im Gleitkomma-Modus ausgeführt!)
  - „%“: Modulo
  - „++“, „- -“: Inkrement- und Dekrement-Operatoren (als „*Prä*“- oder „*Post*“-Operatoren mit gleicher Syntax und Funktionalität wie in PHP oder C)

Operatoren

- String-Operatoren:
  - „+“: Zusammenhängen von Strings (*string concatenation*):

```
"Hello"+" "+"world";  // "Hello world"
```

  - Wenn einer der beiden Operanden des „+“-Operators ein String ist, werden beide in Strings umgewandelt:

```
a = "1" + 2;       // a = "12"
b = 1 + "2";       // b = "12"
c = "1" + 2 + 3;   // c = "123"
d = 1 + 2 + "3";   // d = "33"
```

Operatoren

- Logische Operatoren:
  - „&&“: Logisches „UND“
  - „||“: Logisches „ODER“
  - „!“: Logisches „NICHT“

Operatoren

- Bitweise Operatoren:
  - „&“: Bitweises „UND“
  - „|“: Bitweises „ODER“
  - „^“: Bitweises „EXKLUSIV-ODER“
  - „!“: Bitweises „NICHT“
  - „<<“: Bitweise Linksverschiebung
  - „>>“: Bitweise Rechtsverschiebung mit Vorzeichenerhalt
  - „>>>“: Bitweise Rechtsverschiebung, links immer Null-Bits eingefügt

Operatoren

- Vergleichs-Operatoren (auch für Strings!):
  - Ungleiche Datentypen werden vor dem Vergleich in Zahlen umgewandelt (➔ JavaScript-Dokumentation)
  - „==“, „!=“: Gleich, ungleich
  - „===“, „!==“: Identisch, nicht identisch (wie PHP)
  - „<“, „<=“: Kleiner, kleiner – gleich
  - „>“, „>=“: Größer, größer – gleich

Operatoren

- Zuweisungs-Operatoren:
  - „=“: Gewöhnliche Zuweisung
  - „*op*=“: Zuweisung mit Operation, z.B.:
    ```
    a += b;    // äquivalent zu a = a + b;
    ```
    funktioniert für alle binären arithmetischen und bitweisen Operatoren

Operatoren

- Sonstige Operatoren:
  - „`?:`“: Bedingter Operator (wie in PHP), z.B.:
    ```
    a = x > 0 ? x * y : -x * y;
    ```
  - „`typeof`“: Gibt die Datentype einer Variablen als String an (➔ JavaScript-Dokumentation)
  - „`new`“: Erstellt ein neues Objekt durch Aufruf seines Konstruktors (➔ JavaScript-Dokumentation), z.B. :
    ```
    d = new Date();
    ```

Operatoren

- Sonstige Operatoren:
  - „`delete`“: Löscht *Properties* von Objekten oder Feldelemente (*nicht* die Objekte selbst oder das ganze *Array*!)
  - „`void`“: Für spezielle Aufgabenstellungen, wo die Nebenwirkungen, aber nicht das Ergebnis eines Ausdrucks interessieren (➔ JavaScript-Dokumentation)
  - „`,`“ (Komma): Zum „Aneinanderhängen“ voneinander unabhängiger Befehle

Operatoren

- Feld- und Objektzugriffs-Operatoren:
  - „`[]`“: Auswahl eines Feldelements oder einer *Property* eines Objekts
  - „`.`“: Nur für Auswahl von Objekt-*Properties*
  - Bei Auswahl von Feldelementen muss ein *ganzzahliger numerischer Wert* in den eckigen Klammern stehen
  - Bei Auswahl einer Objekt-*Property* mit „`[]`“ muss ein *String* (der Name der *Property*) in den Klammern stehen

Operatoren

- Feld- und Objektzugriffs-Operatoren:
  - Die folgenden Konstrukte sind äquivalent:
    - `document.lastModified` und `document["lastModified"]`
    - `frames[0].length` und `frames[0]['length']`
  - Die Variante mit „`[]`“ hat den Vorteil, dass der Name der gewählten *Property* dynamisch generiert werden kann

Eigenschaften von JavaScript

- Allgemeines
- Datentypen
- Variable
- Operatoren
- **Konstrukte**
- Funktionen
- JavaScript und HTML

Konstrukte

- Ausdrücke können (wie in PHP oder C) mit „{…}“ gruppiert werden
- Programmverzweigungen:
  - `if`, `else`, `else if`, `switch` mit gleicher Syntax und Funktionalität wie in PHP oder C
  - `break` mit ähnlicher Syntax und gleicher Funktionalität wie in PHP (Unterschiede ➔ JavaScript-Dokumentation)

Konstrukte

- Schleifen:
  - `while`, `do … while`, `for` mit gleicher Syntax und Funktionalität wie in PHP oder C
  - `continue` mit ähnlicher Syntax und gleicher Funktionalität wie in PHP (Unterschiede ➔ JavaScript-Dokumentation)

Konstrukte

- Schleifen:
  - `for/in`: Vergleichbar mit `foreach`-Schleifen in PHP; Beispiel (gibt eine Liste aller Properties `x` eines Objekts `obj` und ihrer Werte `obj[x]` aus):

```
var msg = "Properties von " + obj.name
   + ":\n";
for (var x in obj)
   msg += x + " = " + obj[x] + "\n";
alert (msg);
```


Konstrukte

- Vereinfachter Zugriff auf verschachtelte Objekt-*Properties* – `with`:
  - Beispiel (setzt die Formularelemente `name`, `adresse` und `email` des Formulars `test` zurück):

```
with (frames[1].document.forms.test)
{
   name.value = "";
   adresse.value = "";
   email.value = "";
}
```

Konstrukte

- Alternative:
  - Beispiel (gleiche Funktionalität wie vorhin):

```
var f = frames[1].document.forms.test;

f.name.value = "";
f.adresse.value = "";
f.email.value = "";
```


Eigenschaften von JavaScript

- Allgemeines
- Datentypen
- Variable
- Operatoren
- Konstrukte
- **Funktionen**
- JavaScript und HTML

Funktionen

- Funktionen:
  - Können optional mit dem Befehl `return` ein Ergebnis zurückgeben
  - Syntax wie in PHP oder C

Funktionen

- Definition von Funktionen:
  - Mit dem Schlüsselwort `function`:

```
function mal (x, y)
{
   return x*y;
}
```

  - Mit dem `Function`-Konstruktor:

```
var mal = new Function ("x", "y",
  "return x*y");
```

Funktionen

- Definition von Funktionen:
  - Definition als Funktions-Konstante:

```
var mal = function (x, y)
   {return x*y};
```

  - Gewöhnliche Definition in allen Versionen von JavaScript gültig, `Function`-Konstruktor ab JavaScript 1.1, Funktions-Konstante ab JavaScript 1.2 (kein echtes Kompatibilitäts-Problem – Browser seit ca. 2002 unterstützen JavaScript 1.3)

Funktionen

- Funktionen:
  - Funktionen können als Daten manipuliert werden:

```
a = mal;
b = a (3, 4);
  // äquivalent zu "b = mal (3, 4);"
```

  - Funktionen und ihre Argumente können immer als Methoden bzw. *Properties* eines Objekts behandelt werden

Eigenschaften von JavaScript

- Allgemeines
- Datentypen
- Variable
- Operatoren
- Konstrukte
- Funktionen
- **JavaScript und HTML**

JavaScript und HTML

- JavaScript-Code in HTML eingebettet:
  - Zwischen „`<script>`“ und „`</script>`“-Tags
  - In externer Datei, referenziert durch „`src=`“ oder „`archive=`“-Attribut eines „`<script>`“-Tags
  - In HTML-Event Handler-Attributen (z.B. „`onClick=`“)
  - In einer „`javascript:`“-URL
  - In einem Style Sheet zwischen „`<style type="text/javascript">`“ und „`</style>`“

JavaScript und HTML

- Weitergehende Informationen ➔ JavaScript-Dokumentation
- Manche (sehr) alte Browser (sowie sehr primitive Web-Bots) kennen den `<script>`-Tag (noch) nicht – Tags werden zwar ignoriert, aber JavaScript-Code dazwischen angezeigt – Abhilfe: HTML-Kommentar:

JavaScript und HTML

- Beispiel für JavaScript-Code-Definition:

```
<script language="JavaScript">
<!--

// Beliebiger JavaScript-Code

// -->
</script>
```

Client-seitige Operationen mit JavaScript

- Anwendungsbereiche von JavaScript
- Grundlegende Eigenschaften von JavaScript
- **Formulare und JavaScript**
- Javascript-*Event Handlers*
- AJAX (Asynchronous JavaScript and XML)
- JavaScript-Frameworks
- JavaScript am Client-Rechner

Formulare und JavaScript

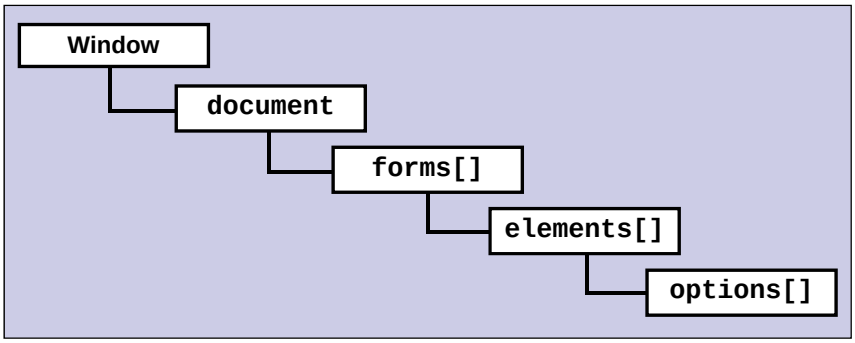
- Objekt-Hierarchie in Client-seitigem JavaScript
- Zugriff auf Formularelemente unter JavaScript
- *Properties* von Formularelementen
- Methoden von Formularen
- Wichtige Funktionen in Formularen
- Beispiel

Formulare und JavaScript

- **Objekt-Hierarchie in Client-seitigem JavaScript**
- Zugriff auf Formularelemente unter JavaScript
- *Properties* von Formularelementen
- Methoden von Formularen
- Wichtige Funktionen in Formularen
- Beispiel

Objekt-Hierarchie

- Vereinfacht – nur Formular-Elemente



Objekt-Hierarchie

- Window – aktuelles Fenster
- `document` – das im aktuellen Fenster angezeigte Dokument
- `forms[]` – Array der im aktuellen Dokument enthaltenen Formulare
- `elements[]` – Array der im aktuellen Formular enthaltenen Formularelemente
- `options[]` – Array der Options-Objekte einer `<select>`-Liste

Objekt-Hierarchie

- Beispiel:

```
<form name="MyForm">
<input type="text" name="Text1">
<select name="Auswahl">
  <option value="eins">Option 1</option>
  <option value="zwei">Option 2</option>
  <option value="drei">Option 3</option>
</select>
</form>
```

Objekt-Hierarchie

- Text „`Option 3`“ kann daher angesprochen werden als:
  `document.forms[0].elements[1].options[2].text`
  - Erstes Formular (Index „`0`“)
  - Zweites Element (Index „`1`“)
  - Dritte Option (Index „`2`“)

Objekt-Hierarchie

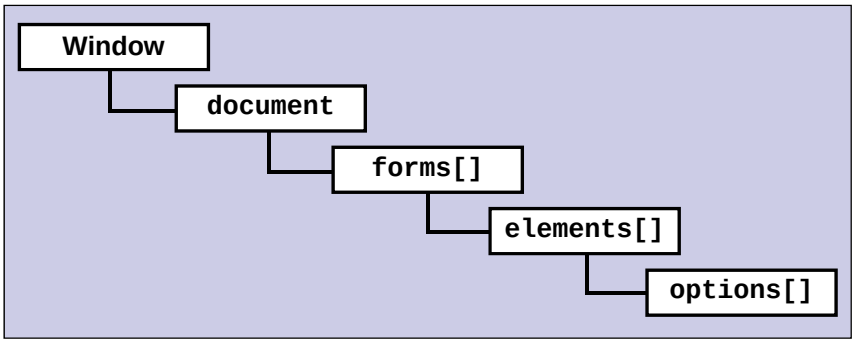
- Spezielle Aspekte beim Zugriff auf Objekte in einem anderen Frame oder einem *Child Window* ➔ JavaScript-Dokumentation

Formulare und JavaScript

- Objekt-Hierarchie in Client-seitigem JavaScript
- **Zugriff auf Formularelemente unter JavaScript**
- *Properties* von Formularelementen
- Methoden von Formularen
- Wichtige Funktionen in Formularen
- Beispiel

Zugriff auf Formularelemente

- Vereinfachte Objekt-Hierarchie



Zugriff auf Formularelemente

- Beispiel:

```
<form name="MyForm">
<input type="text" name="Text1">
<select name="Auswahl">
  <option value="eins">Option 1</option>
  <option value="zwei">Option 2</option>
  <option value="drei">Option 3</option>
</select>
</form>
```


Zugriff auf Formularelemente

- Text „`Option 3`“ kann angesprochen werden als:
  `document.forms[0].elements[1].options[2].text`

Zugriff auf Formularelemente

- Numerische Indizes sind unhandlich und unübersichtlich
- Daher legt JavaScript bei benannten Formularen und Formularelementen zusätzliche *Properties* der `document`-, `forms[]`- und `elements[]`-Objekte an, die den Namen des Formulars bzw. des Formularelements tragen:

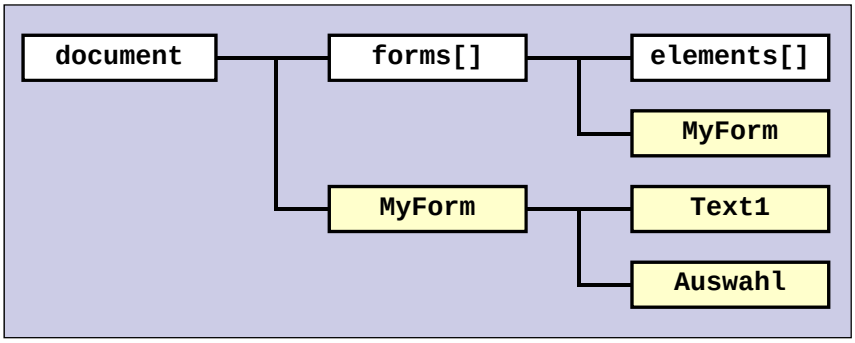
Zugriff auf Formularelemente

- Beispiel:

```
<form name="MyForm">
<input type="text" name="Text1">
<select name="Auswahl">
  <option value="eins">Option 1</option>
  <option value="zwei">Option 2</option>
  <option value="drei">Option 3</option>
</select>
</form>
```

Zugriff auf Formularelemente

- Zusätzliche Properties für benannte Elemente



Zugriff auf Formularelemente

- Text „`Option 3`“ kann daher alternativ angesprochen werden als:
  - `document.forms[0].elements[1].options[2].text`
  - `document.forms['MyForm'].elements['Auswahl'].options[2].text`
  - `document.forms.MyForm.elements.Auswahl.options[2].text`

Zugriff auf Formularelemente

- Text „`Option 3`“ kann daher alternativ angesprochen werden als (Fortsetzung):
  - `document.forms.MyForm.elements.Auswahl.options[2].text`
  - `document.MyForm.Auswahl.options[2].text`
  - `document.MyForm.Auswahl[2].text`

Formulare und JavaScript

- Objekt-Hierarchie in Client-seitigem JavaScript
- Zugriff auf Formularelemente unter JavaScript
- ***Properties* von Formularelementen**
- Methoden von Formularen
- Wichtige Funktionen in Formularen
- Beispiel

Properties von Formularelementen

- `<input type="text">`,
  `<input type="password">`,
  `<input type="hidden">`:
  - `type`: Entsprechend dem „`type=`“-Attribut „`text`“, „`password`“ oder „`hidden`“
  - `name`: Wert des „`name=`“-Attributs
  - **`value`**: Inhalt des Formularelements (bzw. Wert des „`value=`“-Attributs)

Properties von Formularelementen

- `<input type="submit">`,
  `<input type="reset">`,
  `<input type="button">`:
  - `type`: Entsprechend dem „`type=`“-Attribut „`submit`“, „`reset`“ oder „`button`“
  - `name`: Wert des „`name=`“-Attributs
  - `value`: Text des *Button*s (d.h. Wert des „`value=`“-Attributs)

Properties von Formularelementen

- `<input type="checkbox">`:
  - `type`: „`checkbox`“
  - `name`: Wert des „`name=`“-Attributs
  - `value`: Wert des „`value=`“-Attributs
  - **`checked`**: `true`, wenn Checkbox aktiv (Häkchen), sonst `false`

Properties von Formularelementen

- `<input type="radio">`:
  - `type`: „`radio`“
  - `name`: Wert des „`name=`“-Attributs
  - `value`: Wert des „`value=`“-Attributs
  - **`checked`**: `true`, wenn *Radio Button* „gedrückt“, sonst `false`

Properties von Formularelementen

- `<input type="radio">`:
  - Bei mehreren Formularelementen mit gleichem Namen (typisch bei *Radio Buttons*) legt JavaScript ein *Array* mit dem Namen des Formularelements an
  - Wenn kein Element des *Array*s explizit mit „`[]`“ spezifiziert wird, gibt die `value`-*Property* den Wert für die selektierte Option an

Properties von Formularelementen

- `<select>`, `<select multiple>`:
  - *Properties* des Formularelements selbst:
    - `type`: „`select-one`“ bzw. „`select-multiple`“
    - `name`: Wert des „`name=`“-Attributs
    - **`selectedIndex`**: Index der (ersten) selektierten Option; `-1`, wenn keine Option selektiert ist (nur bei „`select-multiple`“)

Properties von Formularelementen

- `<select>`, `<select multiple>`:
  - Properties des `i`-ten Elements des `options[]`-Arrays:
    - **`value`**: Wert des „`value`“-Attributs des `i`-ten „`<option>`“-*Tag*s im „`<select>`“-Konstrukt
    - **`text`**: Text zwischen dem `i`-ten „`<option>`“ – „`</option>`“-*Tag*-Paar im „`<select>`“-Konstrukt
    - **`selected`**: `true`, wenn Option selektiert, sonst `false`

Properties von Formularelementen

- `<textarea>`:
  - `type`: „`textarea`“
  - `name`: Wert des „`name`“-Attributs
  - **`value`**: Inhalt der `<textarea>`

Properties von Formularelementen

- `<input type="file">`:
  - `type`: „`file`“
  - `name`: Wert des „`name=`“-Attributs
  - **`value`**: Eingetragener Dateiname und -pfad oder Daten, die mit dem „Datei öffnen“-Dialog ausgewählt wurden; kann nur gelesen, aber nicht gesetzt werden!
  - Ältere Browser (bis etwa Firefox 1.x) erlaubten noch, über die `value`-Property den gesamten Pfad der hochzuladenden Datei zu ermitteln. Neuere Browser liefern als `value`-Property nur mehr den Dateinamen – Sicherheits-Feature!

Formulare und JavaScript

- Objekt-Hierarchie in Client-seitigem JavaScript
- Zugriff auf Formularelemente unter JavaScript
- *Properties* von Formularelementen
- **Methoden von Formularen**
- Wichtige Funktionen in Formularen
- Beispiel

Methoden von Formularen

- Zwei Methoden von Formularen:
  - `submit()` – Sendet das Formular ab (äquivalent zur Betätigung des *Submit-Button*s)
  - `reset()` – Setzt das Formular zurück (äquivalent zur Betätigung des *Reset-Button*s)
- JavaScript-Code kann damit ein Formular (z.B. zur serverseitigen Aktualisierung von Daten) automatisch abschicken

Methoden von Formularen

- Beispiel:

```
<form name="MyForm">
<input type="text" name="Text1">
<select name="Auswahl">
  <option value="eins">Option 1</option>
  <option value="zwei">Option 2</option>
  <option value="drei">Option 3</option>
</select>
</form>
```

Methoden von Formularen

- Beispiel (Fortsetzung):

```
<script language="JavaScript">

// beliebiger JavaScript-Code

document.MyForm.submit();
  //schickt Formular MyForm ab

</script>
```

Formulare und JavaScript

- Objekt-Hierarchie in Client-seitigem JavaScript
- Zugriff auf Formularelemente unter JavaScript
- *Properties* von Formularelementen
- Methoden von Formularelementen
- **Wichtige Funktionen in Formularen**
- Beispiel

Funktionen in Formularen

- `alert()`:
  - `alert()`: Gibt sein (*String*)-Argument in einem Pop-Up-Fenster aus; Fenster kann nur (mit „OK“) weggeklickt werden:

Funktionen in Formularen

- `alert()`:
  - Programmcode für obiges Beispiel (aus einem onSubmit-Event Handler):

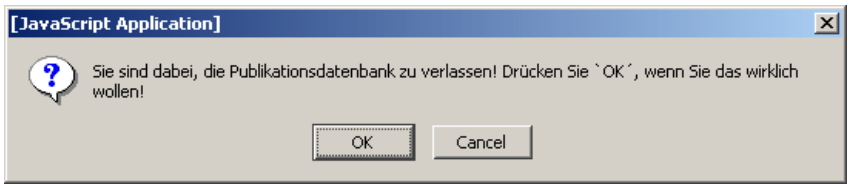
```
if (document.EditForm.titel.value == "")
{
  alert("Sie müssen das Feld `Titel´ angeben!");
  return false;
}
```

Funktionen in Formularen

- `alert()`:
  - `alert()` gibt kein Ergebnis an die aufrufende Funktion zurück
  - Typische Anwendungen:
    - Fehlermeldungen
    - Test-Ausgaben (Argument von `alert()` kann beliebige Ausdrücke beinhalten)

Funktionen in Formularen

- `confirm()`:
  - `confirm()`: Gibt sein (*String*)-Argument in einem Pop-Up-Fenster aus:



Funktionen in Formularen

- `confirm()`:
  - Programmcode für obiges Beispiel (Zielfunktion einer JavaScript-URL):

```
function logout_query ()
{
  if (confirm ("Sie sind dabei, die
    Publikationsdatenbank zu verlassen!
    Drücken Sie `OK´, wenn Sie das wirklich
    wollen!"))
      document.logoutform.submit();
}
```


Funktionen in Formularen

- `confirm()`:
  - `confirm()` gibt bei Klicken auf „OK“ (Default!) den Wert `true`, und bei Klicken auf „Cancel“ den Wert `false` zurück
  - Typische Anwendung: Warnung der User vor potenziell folgenschweren Aktionen

Formulare und JavaScript

- Objekt-Hierarchie in Client-seitigem JavaScript
- Zugriff auf Formularelemente unter JavaScript
- *Properties* von Formularelementen
- Methoden von Formularelementen
- Wichtige Funktionen in Formularen
- **Beispiel**

Beispiel

- Demo-Programme für die Anzeige der Inhalte und *Properties* von Formularelementen sowie Beispiele zu diversen Anwendungen von JavaScript, und Listings der Demo-Programme : Siehe
  `http://karl.riedling.at/index.php?page=366026demos`

Client-seitige Operationen mit JavaScript

- Anwendungsbereiche von JavaScript
- Grundlegende Eigenschaften von JavaScript
- Formulare und JavaScript
- **Javascript-*Event Handlers***
- AJAX (Asynchronous JavaScript and XML)
- JavaScript-Frameworks
- JavaScript am Client-Rechner

Javascript-*Event Handlers*

- Allgemeines zu *Event Handlers*
- *Event Handlers* für Formularelemente
- *Event Handlers* für Formulare
- „Verriegelung“ von Formularen

Javascript-*Event Handlers*

- **Allgemeines zu *Event Handlers***
- *Event Handlers* für Formularelemente
- *Event Handlers* für Formulare
- „Verriegelung“ von Formularen

Allgemeines zu *Event Handlers*

- *Event Handlers* sind JavaScript-Funktionen, die beim Auftreten eines bestimmten Ereignisses („*Events*“) in der Benutzeroberfläche des Browsers ausgeführt werden

Allgemeines zu *Event Handlers*

- Wichtigste *Event Handlers* für Formulare
  - Ändern eines Formularelements – `onChange`
  - Mausklick – `onClick`
  - Tastendruck – `onKeyDown`, `onKeyUp`, `onKeyPress`
  - Element bekommt bzw. verliert den Fokus – `onFocus`, `onBlur`
  - Formular wird abgeschickt bzw. zurückgesetzt – `onSubmit`, `onReset`

Allgemeines zu *Event Handlers*

- Weiterer wichtiger *Event Handler*
  - Laden der Seite abgeschlossen – `onLoad`

Allgemeines zu *Event Handlers*

- Aufruf eines *Event Handlers*:
  - Im HTML-Code über ein (gleichnamiges) HTML-Attribut, z.B.:

```
<input type="text" name="Datum"
onChange="check_datum();">
```


Allgemeines zu Event Handlers

- *Event Handler*-Aufrufe können enthalten:
  - Aufrufe benutzerdefinierter Funktionen:

```
<input type="text" onChange="MyFunc();">
<!– ruft bei Änderung des Textfelds Funktion
MyFunc() auf -->
```

  - Aufrufe beliebiger JavaScript-Funktionen:

```
<input type="text" onChange="submit();">
<!– schickt das Formular bei Änderung des
Textfelds ab -->
```

Allgemeines zu Event Handlers

- *Event Handler*-Aufrufe können enthalten
  - Beliebigen JavaScript-Code (auch mehrere Befehle, aber alles in doppelten Anführungszeichen!):

```
<input type="text" onChange="myvar++;
submit();">
<!– inkrementiert die Variable myvar
bei Änderung des Textfelds und schickt
anschließend das Formular ab -->
```


Allgemeines zu *Event Handlers*

- Die JavaScript-*Event Handler*-Methoden werden ***immer in Kleinbuchstaben*** geschrieben; in HTML hat sich die Schreibweise mit einzelnen Großbuchstaben eingebürgert:
  - HTML: `onClick`
  - JavaScript-Funktion: `onclick()`

Allgemeines zu *Event Handlers*

- Ein Rückgabewert des *Event Handlers* von `false` verhindert Aktionen des Formularelements, für das er definiert wurde – Abschicken oder Zurücksetzen eines Formulars kann z.B. damit unterbunden werden

Javascript-*Event Handlers*

- Allgemeines zu *Event Handlers*
- ***Event Handlers* für Formularelemente**
- *Event Handlers* für Formulare
- „Verriegelung“ von Formularen

Event Handlers für Elemente

- Generell gültig:
  - `onFocus`: Element erhält den Eingabe-Fokus
  - `onBlur`: Element verliert den Eingabe-Fokus

Event Handlers für Elemente

- Bei Texteingabefeldern gültig:
  - `onKeyDown`: Taste wird gedrückt
  - `onKeyUp`: Taste wird losgelassen
  - `onKeyPress`: Ein Tastendruck ist erfolgt (bei *Auto-Repeat* einer Taste gibt es für jede Wiederholung `onKeyPress`, aber je nach Browser unterschiedlich oft `onKeyDown` und `onKeyUp`)
  - Nur manche Browser generieren bei Tabulator usw. oder für *Button*s `onKey…`-Events

Event Handlers für Elemente

- `<input type="text">`,
  `<input type="password">`:
  - `onChange`: Bei allen Browsern
  - `onClick`: Nur bei manchen Browsern

Event Handlers für Elemente

- `<input type="submit">`,
  `<input type="reset">`,
  `<input type="button">`:
  - `onClick`: **Vorsicht:** `onClick` für einen *Submit-Button* ist nicht äquivalent zu `onSubmit` für das Formular! (Gilt analog für *Reset-Button*s)

Event Handlers für Elemente

- `<input type="checkbox">`, `<input type="radio">`:
  - `onClick`: Bei allen Browsern
  - `onChange`: Nur bei manchen Browsern
- `<select>`, `<select multiple>`:
  - `onChange`: Bei allen Browsern
  - `onClick`: Nur bei manchen Browsern

Event Handlers für Elemente

- `<textarea>`:
  - `onChange`: Bei allen Browsern
  - `onClick`: Nur bei manchen Browsern
- `<input type="file">`:
  - `onChange`: Bei den meisten Browsern
  - `onClick`: Bei den meisten Browsern

Javascript-*Event Handlers*

- Allgemeines zu *Event Handlers*
- *Event Handlers* für Formularelemente
- ***Event Handlers* für Formulare**
- „Verriegelung“ von Formularen

Event Handlers für Formulare

- Aufgerufen im `<form>`-Tag; Beispiel:
  `<form name="test" onSubmit="Check();">`
- `onSubmit`: Aufgerufen, wenn das Formular abgeschickt wird
  - Muss nicht durch Mausklick auf *Submit-Button* erfolgt sein; viele Browser schicken Formulare auch beim Drücken der Eingabetaste ab!
- `onReset`: Aufgerufen beim Klicken auf den *Reset-Button*

Javascript-*Event Handlers*

- Allgemeines zu *Event Handlers*
- *Event Handlers* für Formularelemente
- *Event Handlers* für Formulare
- **„Verriegelung“ von Formularen**

„Verriegelung“ von Formularen

- Schutz vor verfrühtem (Seite noch nicht fertig geladen) oder mehrmaligem Abschicken des Formulars
  - Die meisten Browser erlauben das Abschicken eines Formulars, wenn die Seite (selbst das aktuelle Formular) erst zum Teil geladen ist – Gefahr eines Submits mit unvollständigen Daten
  - Die meisten aktuellen Browser zeigen eine Formularseite auch nach dem Abschicken (bis zum Aufbau der neuen Seite) weiter an und erlauben mehrmaliges Abschicken – Mehrfach-Aufrufe der gleichen Funktion

„Verriegelung“ von Formularen

- Beispiel:

```
<head>
<script language="JavaScript">
var ok = 0;
// onSubmit-Event-Handler:
function dosubmit ()
{
  if (! ok) return false;
  ok = 0;
  // weiterer Code
}
</script>
```


„Verriegelung“ von Formularen

- Beispiel (Fortsetzung):

```
</head>
<body onLoad="ok = 1;">
<!--
  HTML-Code mit beliebig vielen
  Formularen
-->
</body>
```

„Verriegelung“ von Formularen

- Funktion:
  - `onLoad`-*Event Handler* im „<body>“-Tag wird ausgeführt, wenn die Seite fertig geladen ist und setzt die Hilfsvariable `ok` auf `1`
  - Bis dahin ist kein *Submit* möglich (`ok == 0` – `onSubmit`-*Event Handler* gibt `false` zurück)
  - Beim ersten *Submit* wird `ok` wieder auf `0` gesetzt, damit weitere *Submit*s verhindert

Client-seitige Operationen mit JavaScript

- Anwendungsbereiche von JavaScript
- Grundlegende Eigenschaften von JavaScript
- Formulare und JavaScript
- Javascript-*Event Handlers*
- **AJAX (Asynchronous JavaScript and XML)**
- JavaScript-Frameworks
- JavaScript am Client-Rechner

AJAX (Asynchronous JavaScript and XML)

- **Allgemeines zu AJAX**
- HTTP-Requests mit AJAX
- Anzeige der Rückgabe-Daten auf der aktuellen Seite

Allgemeines zu AJAX

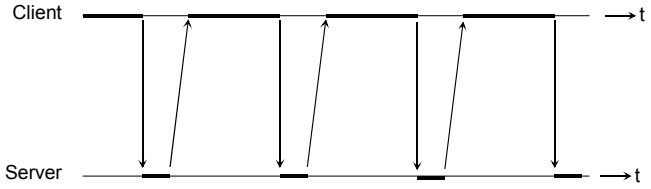
- „Klassische“ HTML-Applikationen: Änderungen des Seiteninhaltes erfordern das Laden einer neuen Seite
- Nachteile:
  - Neuaufbau einer Seite erfordert die (neuerliche) Übertragung aller Inhalte der Seite – hoher Datenverkehr und hohe Server-Belastung
  - Neuaufbau einer Seite wird von manchen Usern als störend empfunden – Verhalten bei Änderungen des Inhalts einer Seite ähnlich zu Desktop-Applikationen angestrebt (Event-getriebene Seiten-Aktualisierung)

Allgemeines zu AJAX

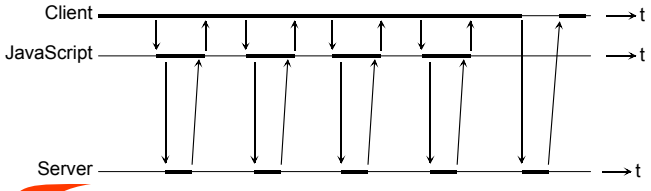
- AJAX – **A**synchronous **J**avaScript **a**nd **X**ML: Asynchrone Datenübertragung zwischen Server und Client
- Zwischen Benutzeroberfläche des Browsers und Server wird Client-seitig eine zusätzliche „Ebene“ eingezogen
- „Ebene“ = System von JavaScript-Klassen, inzwischen Teil des JavaScript-Standards
- Beliebig viele AJAX-Kommunikationen können von einer Seite aus abgesetzt werden, ohne dass die Seite neu geladen zu werden braucht

Allgemeines zu AJAX

- „Klassische“ HTTP-Kommunikation:



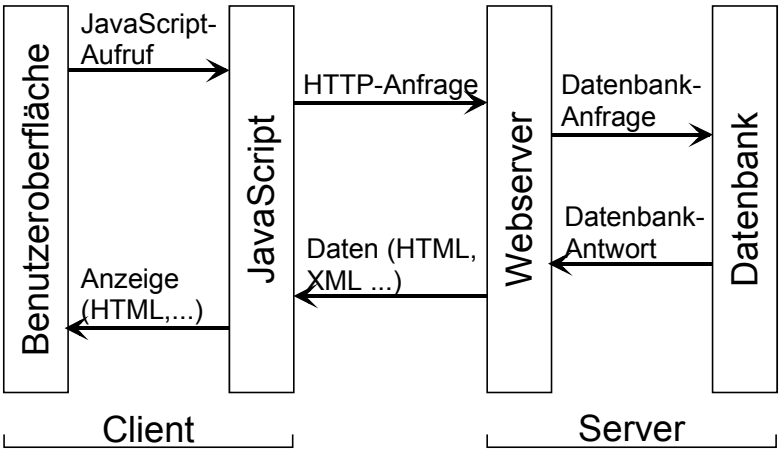
- AJAX-Kommunikation:



Allgemeines zu AJAX

- Ablauf einer AJAX-Kommunikation:
  - Aufruf einer JavaScript-Funktion (typisch durch Event-Handler) → Erstellung eines AJAX-Requestobjekts
  - JavaScript sendet HTTP-Request an aktive Seite (z.B. PHP-Seite) auf dem Server
  - JavaScript empfängt Antwort des Servers auf den Request (als HTTP-, XML- oder sonstige Inhalte)
  - JavaScript zeigt Inhalte innerhalb der Seite an

Allgemeines zu AJAX



Allgemeines zu AJAX

- Voraussetzungen für AJAX:
  - HTML oder XHTML
  - Ein *Document Object Model* zur Repräsentation der Inhalte
  - JavaScript zur Manipulation des *Document Object Models* und zur dynamischen Darstellung der Inhalte
  - Das *XMLHttpRequest*-Objekt, das im Browser definiert sein muss

Allgemeines zu AJAX

- Verfügbarkeit von AJAX:
  - Direkte Verwendung des *XMLHttpRequest*-Objekts oder äquivalenter Objekte in (älteren) anderen Browsern – sehr Browser-abhängig
  - Unter Verwendung diverser Frameworks (jQuery, Prototype,…) – weitaus portabler!
  - Prototype: entwickelt ab 2005 von Sam Stephenson, seit 2010 von Andrew Dupont – `prototype.js` (verfügbar über `http://www.prototypejs.org/`), neueste Release: Version 1.7.3 (September 2015)

Allgemeines zu AJAX

- AJAX-Unterstützung gängiger Browser mit direkter Verwendung des *XMLHttpRequest*-Objekts:
  - *Gecko*-Browser: *Firefox*, *SeaMonkey* ab Version 2
  - *Internet Explorer*: ab Version 7; viele Features erst ab Version 10
  - *Opera*: ab Version 9; viele Features erst mit Versionen > 12
  - *Safari*: ab Version 1.2

Allgemeines zu AJAX

- AJAX-Unterstützung gängiger Browser mit `prototype.js`:
  - *Gecko*-Browser (*Netscape*, *Mozilla*, *Firefox*, *SeaMonkey*): Alle ab *Netscape* 6.0 (!)
  - *Internet Explorer*: ab Version 5.5
  - *Opera*: Volle Funktionalität mit der Standard-Implementierung der AJAX-Klassen erst ab Version 9 (Fehlermeldung, aber korrekte Funktion mit *Opera* 8)
  - *Safari*: ab Version 1.2

Allgemeines zu AJAX

- Varianten von AJAX:
  - Aufruf von Ressourcen:
    - REST (**Re**presentational **S**tate **T**ransfer): Aufruf mit Standard-HTTP-Methoden, aber für jede Ressource individueller URI
    - SOAP (ursprünglich „**S**imple **O**bject **A**ccess **P**rotocol“): Austausch XML-basierter Nachrichten unter Verwendung des HTTP-Protokolls, typisch für Webservices verwendet

Allgemeines zu AJAX

- Varianten von AJAX:
  - Datentransfer vom Server zum *XMLHttpRequest*-Objekt :
    - REST-artige Verfahren (Text-Daten)
    - JSON („Jason“ – **J**ava**S**cript **O**bject **N**otation): JavaScript-spezifisches Format (wird auch von PHP unterstützt!)
    - SOAP – XML-Daten
    - Diverse proprietäre XML-Verfahren

AJAX (Asynchronous JavaScript and XML)

- Allgemeines zu AJAX
- **HTTP-Requests mit AJAX**
- Anzeige der Rückgabe-Daten auf der aktuellen Seite

HTTP-Requests mit AJAX

- Beispiel: Verbleibende Zeit in einer Session anzeigen – mit `XMLHttpRequest`:

```
function SessionTime() {
  var AR = new XMLHttpRequest();
  AR.onreadystatechange = function() {
    if (AR.readyState == 4 && AR.status == 200)
      document.getElementById("Zeit").innerHTML
      = AR.responseText;
  }
  AR.open ("GET", "mytime.php?ID=12345678", true);
  AR.send();
  window.setTimeout('SessionTime()', 60000);
}
```

HTTP-Requests mit AJAX

- Objekt der Klasse `XMLHttpRequest` erstellen
- (Anonyme) Funktion erstellen, die bei Vorliegen der Antwort des Servers die gelieferten Daten Client-seitig verarbeitet:
  - `onreadystatechange`: Function-Objekt von `XMLHttpRequest`
  - `readyState == 4`: Daten vom Server liegen komplett vor
  - `status == 200`: HTTP-Status (vom Server geliefert) ist „Ok“

HTTP-Requests mit AJAX

- Zielseite definieren mit Methode `open()`:
  - Request-Methode („`GET`“, „`POST`“…)
  - Zielseiten-URL (evtl. mit Parametern in üblicher GET-Notation)
  - Optionaler Parameter, „`true`“ (Default) für asynchrone Operation
- Request absenden mit Methode `send()`

HTTP-Requests mit AJAX

- `window.setTimeout('SessionTime()', 60000)`: ruft JavaScript-Funktion `SessionTime()` alle 60000 Millisekunden (also einmal pro Minute) auf – in diesem Fall automatische periodische (nicht interaktive) Aktualisierung
- Eine Funktion mit ähnlicher Struktur wie `SessionTime()` (ohne `window.setTimeout()`) kann aber z.B. auch als Event-Handler-Funktion aufgerufen werden (`onClick=`, `onChange=` …)

HTTP-Requests mit AJAX

- Beispiel: Verbleibende Zeit in einer Session anzeigen – mit `prototype.js`:

```
function SessionTime() {
  var AjaxRequest = new Ajax.Request ("mytime.php",
  {
    method: 'get',
    parameters: 'SessionID=12345678',
    onComplete: show_timeout
  }
  );

  window.setTimeout('SessionTime()', 60000);
}
```

HTTP-Requests mit AJAX

- Konstruktor `Ajax.Request` bewirkt Aufruf einer aktiven Seite (hier PHP – `mytime.php`) mit den folgenden Informationen
  - `method: 'get'`: Übermittlungsmethode (`get` oder `post`) – `get` ist am einfachsten
  - `parameters`: Aufrufparameter (wie bei einer `get`-URL; mehrere Parameter mit "`&`" aneinanderreihen)
  - `onComplete`: Name der JavaScript-Funktion (ohne Klammern und Parameter!), die bei Rückgabe der Daten vom Server ausgeführt werden soll

AJAX (Asynchronous JavaScript and XML)

- Allgemeines zu AJAX
- HTTP-Requests mit AJAX ohne Parameter
- **Anzeige der Rückgabe-Daten auf der aktuellen Seite**

Anzeige der Rückgabe-Daten

- Mit `XMLHttpRequest`: Funktion, die als `XMLHttpRequest.onreadystatechange` registriert wurde:

```
if (AR.readyState == 4 && AR.status == 200)
      document.getElementById("Zeit").innerHTML
      = AR.responseText;
```

  - AR: bei der Erstellung der Anfrage an den Server erstelltes Request-Objekt
  - `document.getElementById('…').innerHTML`: befüllt Seitenelement mit der angegebenen ID (hier `Zeit`) mit den vom Server gelieferten Inhalten

Anzeige der Rückgabe-Daten

- Mit `prototype.js`: Funktion, die mit dem Konstruktor `Ajax.Request` unter `onComplete` registriert wurde:

```
function show_timeout(originalRequest)
{
  document.getElementById('Zeit').innerHTML =
  originalRequest.responseText;
}
```

  - `originalRequest`: beliebiger Platzhalter-Name
  - `document.getElementById('…').innerHTML`: befüllt Seitenelement mit der angegebenen ID (hier `Zeit`) mit den vom Server gelieferten Inhalten

Anzeige der Rückgabe-Daten

- Das zu befüllende Element in der HTML-Seite kann folgendermaßen definiert werden:
  `<div id="Zeit"></div>`
- In gleicher Weise können aber auch Formularelemente mit neuen Inhalten versehen werden
- Andere Möglichkeiten: Steuerung der Sichtbarkeit von Seiteninhalten durch die `onComplete`-Funktion über Styles oder CSS

Client-seitige Operationen mit JavaScript

- Anwendungsbereiche von JavaScript
- Grundlegende Eigenschaften von JavaScript
- Formulare und JavaScript
- Javascript-*Event Handlers*
- AJAX (Asynchronous JavaScript and XML)
- **JavaScript-Frameworks**
- JavaScript am Client-Rechner

JavaScript-Frameworks

- Die Verwendung von JavaScript kann durch die Verwendung geeigneter Frameworks wesentlich erleichtert und von implementierungsspezifischen Besonderheiten der Browser entkoppelt werden.
- Manche Frameworks ermöglichen überhaupt erst komplexe Features (z.B. Animationen, Seiten-Manipulationen, Texteditor-Features)

JavaScript-Frameworks

- Beispiele für JavaScript-Frameworks:
  - jQuery (`https://jquery.com/`): Seitenmanipulation, erweitertes Event-Handling, Animationen, AJAX
  - Prototype (`http://www.prototypejs.org/`): DOM-Erweiterungen, AJAX, JSON, erweitertes Event-Handling
  - TinyMCE (`https://www.tinymce.com/`): WYSIWYG-Editor, basierend auf `<textarea>`-Formularelementen

Client-seitige Operationen mit JavaScript

- Anwendungsbereiche von JavaScript
- Grundlegende Eigenschaften von JavaScript
- Formulare und JavaScript
- Javascript-*Event Handlers*
- AJAX (Asynchronous JavaScript and XML)
- JavaScript-Frameworks
- **JavaScript am Client-Rechner**

JavaScript am Client-Rechner

- Probleme bei der Verwendung von JavaScript
  - Client-Browser unterstützt nicht JavaScript
  - Client-Browser hat JavaScript deaktiviert
  - Client-Browser unterstützt eine unterschiedliche (z.B. niedrigere) Version von JavaScript (alle seit 2002 erschienenen Browser-Versionen unterstützen jedenfalls JavaScript 1.3)

JavaScript am Client-Rechner

- Programmtechnische Maßnahmen:
  - JavaScript überhaupt nicht verwenden
  - Nur fundamentale Funktionen von JavaScript verwenden
  - Seiten funktionieren auch ohne JavaScript
  - Verschiedene Versionen der Seite mit und ohne JavaScript-Unterstützung
  - Dynamische Anpassung der Seiten an aktuelle JavaScript-Version ➔ JavaScript-Dokumentation

JavaScript am Client-Rechner

- Seiten funktionieren auch ohne JavaScript
  - Möglich, wenn JavaScript nur für ergänzende Funktionen verwendet wird (z.B. automatische Anpassung von Selektionen in Formularen, automatische Aktualisierung von Seiten, usw.), die bei Bedarf auch manuell vorgenommen werden können (z.B. *Button* für „Aktualisieren“ vorsehen)

JavaScript am Client-Rechner

- Verschiedene Versionen der Seite
  - Bei Nicht-Vorhandensein von JavaScript (oder der geeigneten Version) kann entweder auf eine andere Version der Seiten verzweigt oder aber der Zutritt zu den folgenden Seiten verweigert werden

JavaScript am Client-Rechner

- Beispiel:

```
<head>
<script language="JavaScript"
type="text/javascript">
<!--
// Wenn der Browser JavaScript kann,
// springt er zum Login, sonst nicht.
window.location.replace("login.php");
//-->
</script>
</head>
```

JavaScript am Client-Rechner

- Beispiel (Fortsetzung):

```
<body>
<b>Sie werden automatisch zum Login
weitergeleitet.<br>
Sollte dies nicht funktionieren, ist
wahrscheinlich JavaScript deaktiviert.<br> Bitte
aktivieren Sie JavaScript oder folgen Sie einem
der untenstehenden Links!
</b>
</body>
```