

JavaScript ist anders

Karlsruher Entwicklertag
23. Juni 2009

Wer ich bin...



Wer ich bin...



Mein eigener Chef

(Extremer) Softwareentwickler

(Agiler) Coach

Testgetrieben

<http://johanneslink.net>

Warum JavaScript?

Warum JavaScript?

"Html 5 and JavaScript are the future of Rich Client Development"

Warum JavaScript?

"Html 5 and JavaScript are the future of Rich Client Development"

...sagt Google und handelt so

Historie

- Von Brendan Eich 1995 für Netscape Navigator 2 unter dem Namen LiveScript entwickelt
- Seit 1997 Standardisierung als ECMAScript
- Großes Revival mit dem Ajax-Hype
- Firefox 3.0 enthält Version 1.8 (ES 3.?)
- Die Zukunft:
ECMAScript 3.1, 4 und „5th edition“

Java?-Script

- Curly Braces
- Syntax des Methodenaufrufs
- Einige Basistypen und Methoden:
String, Date, toString(), ...
- Aber: Semicolons am Zeilenende
optional

Laufzeitumgebung

Laufzeitumgebung

- Im Browser
 - ▶ Enge Verknüpfung mit DOM
 - ▶ Ajax, Timer und Event-Mechanismus
 - ▶ In jedem Browser im Detail unterschiedlich
 - ▶ Beste Performance in Chrome

Laufzeitumgebung

- Im Browser
 - ▶ Enge Verknüpfung mit DOM
 - ▶ Ajax, Timer und Event-Mechanismus
 - ▶ In jedem Browser im Detail unterschiedlich
 - ▶ Beste Performance in Chrome
- Auf dem Server
 - ▶ Via Java's Scripting-Mechanismus
 - ▶ Rhino-JS nativ
 - ▶ Zugriff auf Java-Objekte und -Bibliotheken

Spracheigenschaften

Spracheigenschaften

- Grundkonstrukte
 - ▶ Funktionen
 - ▶ Objekte
 - ▶ Closures

Spracheigenschaften

- Grundkonstrukte
 - ▶ Funktionen
 - ▶ Objekte
 - ▶ Closures
- Dynamische Typisierung

Spracheigenschaften

- Grundkonstrukte
 - ▶ Funktionen
 - ▶ Objekte
 - ▶ Closures
- Dynamische Typisierung
- (Fast) alles ist (zur Laufzeit) offen

Spracheigenschaften

- Grundkonstrukte
 - ▶ Funktionen
 - ▶ Objekte
 - ▶ Closures
- Dynamische Typisierung
- (Fast) alles ist (zur Laufzeit) offen
- Objektorientiert
 - ▶ aber nicht klassenbasiert!

Primitive Typen

```
var name = 'Johannes';  
assertEqual('Johannes Link', name + ' Link');
```

```
var eins = 1;  
assertEqual(2, eins + 1);  
assertEqual('11', eins + '1');
```

```
var liste = [1, 2, 'drei'];  
assertEqual('drei', liste[2]);
```

```
var isBig = eins > 1000;  
if (isBig) fail();
```

Primitive Typen

```
var name = 'Johannes';  
assertEqual('Johannes Link', name + ' Link');
```

```
var eins = 1;  
assertEqual(2, eins + 1);  
assertEqual('11', eins + '1');
```

```
var liste = [1, 2, 'drei'];  
assertEqual('drei', liste[2]);
```

```
var isBig = eins > 1000;  
if (isBig) fail();
```

```
assertEqual('ff', (255).toString(16));  
var text = 'Dies ist ein Text.';  
assertEqual(18, text.length);  
assertEqual('ein Text.', text.substring(9));
```

Funktionen

```
function sayHelloTo(name) {  
    return 'Hello, ' + name  
}  
sayHelloTo('Johannes')
```

Funktionen

```
function sayHelloTo(name) {  
    return 'Hello, ' + name  
}  
sayHelloTo('Johannes')
```

```
var sayByeTo = function(name) {  
    return 'Bye, ' + name;  
}  
sayByeTo('Michael');
```

Funktionen

```
sayHelloTo('Johannes');  
function sayHelloTo(name) {  
    return 'Hello, ' + name;  
}
```

```
var sayByeTo = function(name) {  
    return 'Bye, ' + name;  
}  
sayByeTo('Michael');
```

Funktionen

```
sayHelloTo('Johannes');  
function sayHelloTo(name) {  
    return 'Hello, ' + name;  
}
```

```
var sayByeTo = function(name) {  
    return 'Bye, ' + name;  
}  
sayByeTo('Michael');
```

```
var sayByeTo = function(first, last) {  
    return 'Bye, ' + name + ' ' + last  
}  
assert(sayByeTo('Michael') == ??????)
```

Funktionen

```
sayHelloTo('Johannes');  
function sayHelloTo(name) {  
    return 'Hello, ' + name;  
}
```

```
var sayByeTo = function(name) {  
    return 'Bye, ' + name;  
}  
sayByeTo('Michael');
```

```
var sayByeTo = function(first, last) {  
    return 'Bye, ' + name + ' ' + last  
}  
assert(sayByeTo('Michael') == 'Bye, Michael undefined')
```

Funktionen

```
sayHelloTo('Johannes');  
function sayHelloTo(name) {  
    return 'Hello, ' + name;  
}
```

```
var sayByeTo = function(name) {  
    return 'Bye, ' + name;  
}  
sayByeTo('Michael');
```

```
var sayByeTo = function(first, last) {  
    return 'Bye, ' + arguments[0];  
}  
assert(sayByeTo('Michael', 'Jordan') == 'Bye, Michael');
```

Objekte(1)

```
var johannes = new Object();
johannes.firstName = "Johannes";
johannes.lastName = "Link";
johannes.name = function() {
    return this.firstName + ' ' + this.lastName;
}
assertEqual('Johannes Link', johannes.name());
```

Objekte(1)

```
var johannes = new Object();
johannes.firstName = "Johannes";
johannes.lastName = "Link";
johannes.name = function() {
    return this.firstName + ' ' + this.lastName;
}
assertEqual('Johannes Link', johannes.name());
```

```
var johannes = {
    firstName: 'Johannes', lastName: 'Link',
    name: function() {
        return this.firstName + ' ' + this.lastName;
    }
}
assertEqual('Johannes Link', johannes.name());
```

Objekte (2)

Objekte (2)

```
for(var prop in johannes) {  
    println(prop + ": " + johannes[prop].toString());  
}
```

Objekte (2)

```
for(var prop in johannes) {  
    println(prop + ": " + johannes[prop].toString());  
}
```

```
>> firstName: Johannes  
>> lastName: Link  
>> name: function () {...}
```

Objekte (2)

```
for(var prop in johannes) {  
    println(prop + ": " + johannes[prop].toString());  
}
```

```
>> firstName: Johannes  
>> lastName: Link  
>> name: function () {...}
```

```
johannes.name = function() {  
    return this.firstName.charAt(0) + '. ' + this.lastName;  
}  
assertEqual('J. Link', johannes.name());
```

Funktionen sind Objekte

```
var doppelt = function() {  
    return 2 * doppelt.zahl;  
}  
doppelt.zahl = 3;  
  
assertEqual(6, doppelt());  
assertEqual(6, doppelt.call());
```

Funktionen sind Objekte

```
var doppelt = function() {  
    return 2 * doppelt.zahl;  
}  
doppelt.zahl = 3;  
  
assertEqual(6, doppelt());  
assertEqual(6, doppelt.call());
```

```
doppelt.dreifach = function() {  
    return this() * 3;  
}  
assertEqual(18, doppelt.dreifach());
```

Funktionen sind Objekte

```
var doppelt = function() {  
    return 2 * doppelt.zahl;  
}  
doppelt.zahl = 3;  
  
assertEqual(6, doppelt());  
assertEqual(6, doppelt.call());
```

```
doppelt.dreifach = function() {  
    return this() * 3;  
}  
assertEqual(18, doppelt.dreifach());
```

```
function fuehreAus(eineFunktion) {  
    return eineFunktion();  
}  
assertEqual(6, fuehreAus(doppelt));
```

Anonyme Funktionen

```
(function(mul1, mul2) {  
    return mul1 * mul2;  
})(7, 3);
```

Anonyme Funktionen

```
(function(mul1, mul2) {  
    return mul1 * mul2;  
})(7, 3);
```

```
function fuehreAus(eineFunktion, p1, p2) {  
    return eineFunktion(p1, p2);  
}  
var wert = fuehreAus(function(m1, m2) {  
    return m1 * m2;  
}, 7, 3);  
assertEqual(21, wert);
```

Iteration als typischer Anwendungsfall

Iteration als typischer Anwendungsfall

```
var namen = ["Johannes", "Jens", "Dierk"];  
var auswahl = namen.findAll(function(name) {  
    return name.match("J.*");  
})  
assertEqual(["Johannes", "Jens"], auswahl);
```

Iteration als typischer Anwendungsfall

```
var namen = ["Johannes", "Jens", "Dierk"];  
var auswahl = namen.findAll(function(name) {  
    return name.match("J.*");  
})  
assertEqual(["Johannes", "Jens"], auswahl);
```

```
List<String> auswahl = new ArrayList<String>();  
for (String name : namen) {  
    if (name.matches("J.*")) {  
        auswahl.add(name);  
    }  
}
```

Äquivalenter Java Code

Scope

```
var global = 1;
function fn() {
  var inFn = 2;
  assert(global == 1);
  assert(inFn == 2);
  function other() {
    var inOther = 3;
    assert(inFn == 2);
    assert(inOther == 3);
  }
  other();
  (function() {
    assert(inFn == 2);
    other();
    assert(typeof inOther == 'undefined');
  })()
}

assert(typeof inFn == 'undefined');
fn();
```

with

```
function outerFun() {
    return 'outer';
}

var myScope = {
    innerFun: function() {
        return 'inner';
    },
    innerVar: 4
}

with(myScope) {
    assert(outerFun() == 'outer');
    assert(innerFun() == 'inner');
    assert(innerVar == 4);
    innerVar = 3;
    newVar = 42;
}

assert(myScope.innerVar == 3);
assert(typeof myScope.newVar == 'undefined');
assert(newVar == 42);
```

Context

Context

```
var Fuenf = {  
  malZahl: function(num) {  
    return num * this.zahl;  
  },  
  zahl: 5;  
}  
assertEqual(10, Fuenf.malZahl(2));
```

Context

```
var Fuenf = {  
  malZahl: function(num) {  
    return num * this.zahl;  
  },  
  zahl: 5;  
}  
assertEqual(10, Fuenf.malZahl(2));
```

```
var Sieben = {  
  malZahl: Fuenf.malZahl,  
  zahl: 7  
}  
assertEqual(14, Sieben.malZahl(2));  
assertEqual(10, Sieben.malZahl.call(Fuenf, 2));  
assertEqual(10, Sieben.malZahl.apply(Fuenf, [2]));
```

Closures

Closures

```
var myClosure;  
var outerNum = 3;  
  
(function() {  
    var innerNum = 7;  
    var innerFun = function(num) {  
        return num * innerNum;  
    }  
    myClosure = function() {  
        return innerFun(outerNum);  
    };  
})();
```

```
assertEqual(21, myClosure());  
outerNum = 4;  
assertEqual(28, myClosure());
```

Closures sind Funktionen mit Zugriff auf alle Funktionen und Variablen, die bei Erzeugung der Funktion im aktuellen Scope waren

Closure-Beispiel: Klick-Zähler

```
var countClicks = 0;  
document.addEventListener("click", function() {  
    alert(++countClicks);  
});  
countClicks = 42;
```

Closure-Beispiel: Klick-Zähler

```
var countClicks = 0;
document.addEventListener("click", function() {
    alert(++countClicks);
});
countClicks = 42;
```

```
(function() {
    var countClicks = 0;
    document.addEventListener("click", function() {
        alert(++countClicks);
    });
})();
```

Klasse == Konstruktor == Funktion

```
function Person(firstName, lastName) {  
  this.firstName = firstName;  
  this.lastName = lastName;  
}
```

```
Person.prototype.name = function() {  
  return this.firstName + ' ' + this.lastName;  
}
```

```
var johannes = new Person('Johannes', 'Link');  
assertEqual('Johannes Link', johannes.name());  
assert(johannes instanceof Person);
```

Klasse == Konstruktor == Funktion

```
function Person(firstName, lastName) {  
  this.firstName = firstName;  
  this.lastName = lastName;  
}
```

```
Person.prototype.name = function() {  
  return this.firstName + ' ' + this.lastName;  
}
```

```
var johannes = new Person('Johannes', 'Link');  
assertEqual('Johannes Link', johannes.name());  
assert(johannes instanceof Person);
```

Vererbung

Vererbung

```
function Person(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}
Person.prototype.name = function() {...}

function Student(firstName, lastName, fieldOfStudy) {
  Person.call(this, firstName, lastName);
  this.fieldOfStudy = fieldOfStudy;
}
Student.prototype = new Person();
Student.prototype.describe = function() {
  return this.name() + ' is a student of ' + this.fieldOfStudy;
}

var jannek = new Student('Jannek', 'Link', 'Philosophy');
assert(jannek.describe() == 'Jannek Link is a student of Philosophy');
```

Geerbtes Verhalten

Object.prototype
+ Person.prototype
+ Student.prototype
+ Instance properties

} Live Update

} Always supersedes
prototype properties

= Object properties

aus [John Resig: Secrets of the JavaScript Ninja](#)

Zusammenfassung

- Highlights
 - ▶ Funktionen und Closures als vollwertige Objekte
 - ▶ Prototypen und Konstruktoren statt Klassen
 - ▶ Alles ist zur Laufzeit offen
 - ➔ erlaubt sehr knappen Code

Zusammenfassung

- Highlights

- ▶ Funktionen und Closures als vollwertige Objekte
- ▶ Prototypen und Konstruktoren statt Klassen
- ▶ Alles ist zur Laufzeit offen
- ➔ erlaubt sehr knappen Code

- Nicht behandelt

- ▶ `eval()`
- ▶ Reguläre Ausdrücke
- ▶ Timer und asynchrones Verhalten
- ▶ Event-Mechanismen

Fragen und Anmerkungen?

[http://www.slideshare.net/jlink/
java-script-ist-anders/](http://www.slideshare.net/jlink/java-script-ist-anders/)