

# Advanced JavaScript

Liip Techday 2008

Christian Stocker & Tobias Ebnöther, Liip AG

# Agenda

- ▶ History (very short)
- ▶ Datatypes
- ▶ The Module Pattern
- ▶ Inheritance
- ▶ jslint

# History

- ▶ Developed by Brendan Eich at Netscape
- ▶ For Netscape Navigator 2
- ▶ Instantly became massively popular
- ▶ Never got a trial period
- ▶ Never could be polished based on actual use
- ▶ -> The language is powerful...
- ▶ and flawed ...

# Datatypes

- ▶ How much datatypes are there in javascript?

# Data Types

- ▶ 3 primitive types
  - ▶ Boolean
  - ▶ Number
  - ▶ String
- ▶ 2 special values
  - ▶ null
  - ▶ undefined
- ▶ Everything else
  - ▶ object

# Boolean

- ▶ true
- ▶ false
  - ▶ null
  - ▶ undefined
  - ▶ ""
  - ▶ 0
  - ▶ NaN

# Number

- ▶ 64 bit floating point
- ▶ no integer
- ▶ NaN and Infinity

# String

- ▶ zero or more Unicode Characters (UTF-16)
- ▶ “A” === “\u0041”
- ▶ ‘cat’.toUpperCase() === ‘CAT’

# Objects

- ▶ `var empty_object = {}`
- ▶ `var stooge = {  
    "first-name": "Jerome",  
    "last-name": "Howard"  
}`

# Nested Objects

```
▶ var flight = {  
  airline: "Oceanic",  
  number: 815,  
  departure: {  
    IATA: "SYD",  
    time: "2004-09-22 14:55",  
    city: "Sydney"  
  },  
  arrival {  
    IATA: "LAX",  
    time: "004-09-23 10:42",  
    city: "Los Angeles"  
  }  
};
```

# Objects retrieval

- ▶ `stooge['first-name'] // "Jerome"`
- ▶ `flight.departure.IATA // SYD`
- ▶ `flight.foo // undefined`
- ▶ `var status = flight.status || "unknown";`
- ▶ `flight.equipment.model // TypeError`
- ▶ `flight.equipment && flight.equipment.model`

# Reflection

- ▶ `typeof flight.number // "number"`
- ▶ `typeof flight.status // "string"`
- ▶ `typeof flight.arrival // "object"`
- ▶ `typeof flight.manifest // "undefined"`
- ▶ `typeof flight.toString // "function"`

# Objects loopen

```
▶ for (var n in myHashtable) {  
    if (myHashtable.hasOwnProperty(n))  
        document.writeln("<p>" + n + ": " + myHashtable[n] +  
            "</p>");  
}
```

# Arrays

- ▶ Anders als in PHP!
- ▶ `myList = ['oats', 'peas', 'beans', 'barley'];`  
`emptyArray = [];`  
`month_lengths = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];`

# Namespaces

# Namespaces

- \* Only one global needed
- \* Nicely capsuled
- \* Prevent conflicts
- \* Make excessive use

# YUI Namespaces

\* Handles sub spaces and creates all needed variables

```
YAHOO.namespace ("myproduct2.mysubproject1");  
  
YAHOO.myproduct2.mysubproject1.Class1 = function(info) {  
    alert(info);  
};
```

# Namespaces without YUI

\* Use typeof undefined

```
if ('undefined' == typeof(liip)) {  
    liip = {};  
}  
  
liip.example = {  
    //your stuff  
};
```

# Private Members

# Private Members

- \* Do real private (unaccessible methods)
- \* Prevent ugly `_foo` methods (don't use them anyways!)
- \* Create privileged functions (not changeable)

# Private Members

\* Public

```
function Constructor(...) {  
    this.membername = value;  
}  
Constructor.prototype.membername = value;
```

# Private Members

## \* Private

```
function Constructor(...) {  
    var that = this;  
    var membername = value;  
    function membername(...) {...}  
}
```

# Private Members

\* Privileged

```
function Constructor(...) {  
    this.membername = function (...) {...};  
}
```

# Private Members Example

```
function Container(param) {  
  
    function dec() {  
        secret -= 1;  
        return (secret > 0) ? true : false;  
    }  
  
    this.member = param;  
    var secret = 3;  
    var that = this;  
  
    this.service = function () {  
        if (dec()) {  
            return this.member;  
        } else {  
            return null;  
        }  
    }  
};
```

# Module Pattern

# Module Pattern

- \* The combination of Namespaces and Private Members
- \* Less painful than Private Members pattern
- \* Easily readable

# Module Pattern

## \* Private

```
YAHOO.myProject.myModule = function () {  
  
    var myPrivateVar = "private";  
    var myPrivateMethod = function () {  
        YAHOO.log("I'm private too");  
    }  
  
}();
```

# Module Pattern

\* Public

```
YAHOO.myProject.myModule = function () {  
  
    return {  
        myPublicProperty: "public",  
        myPublicMethod: function () {  
            YAHOO.log("I'm public too");  
        }  
    };  
  
}();
```

# Module Pattern Example

```
YAHOO.myProject.myModule = function () {  
  
    var myPrivateVar = "private";  
    var myPrivateMethod = function () {  
        YAHOO.log("I'm private too");  
    }  
  
    return {  
        myPublicProperty: "public",  
        myPublicMethod: function () {  
            YAHOO.log("I'm public too");  
            YAHOO.log(myPrivateVar);  
            YAHOO.log(myPrivateMethod());  
            YAHOO.log(this.myPublicProperty); // Native scope!  
        }  
    };  
  
}();
```

# Inheritance

- ▶ JavaScript is a class-free, object-oriented language
- ▶ Uses prototypal inheritance instead of classical inheritance

# Pseudoclassical

```
▶ var Mammal = function (name) {  
    this.name = name;  
};  
Mammal.prototype.get_name = function() {  
    return this.name;  
};  
Mammal.prototype.says = function() {  
    return this.saying || "";  
};
```

# Pseudoclassical - Extend

```
▶ var Cat = function (name) {  
    this.name = name;  
    this.saying = "miau";  
};  
Cat.prototype = new Mammal();  
Cat.prototype.get_name = function() {  
    return this.says() + ' ' + this.name + ' ' + this.name;  
}
```

# Pseudoclassical - Extend II

- ▶ `var myCat = new Cat("Henrietta");`
- ▶ `var says = myCat.says(); // "meow"`
- ▶ `var name = myCat.get_name(); // 'meow Henrietta meow'`

# Pseudoclassical mit YUI

- ▶ `YAHOO.lang.extend(class,parentClass)`
- ▶ plus “superclass” (wie parent in PHP)

# Pseudoclassical mit YUI

```
▶ var Cat = function (name) {  
    Cat.superclass.constructor.call(this,name);  
    this.saying = "miau";  
}  
YAHOO.lang.extend(Cat, Mammal);  
...
```

# YAHOO.lang.augment (aggregate)

```
▶ var Mammal = function() {};  
  Mammal.prototype.says = function() {  
    return this.saying || "";  
  }  
var swimFeature = function() {};  
swimFeature.prototype.swim = function() {  
  alert("swim");  
}  
var Dolphin = function() {}
```

```
YAHOO.lang.extend(Dolphin, Mammal);  
YAHOO.lang.augment(Dolphin, swimFeature);  
myDolphin = new Dolphin();  
myDolphin.saying = "blubb";  
// http://trash.chregu.tv/advjs2.html
```

# Prototypal Inheritance

- ▶ Keine Klassen
- ▶ Fokus auf Objekten
- ▶ Neues Objekt erbt Eigenschaften von “altem” Objekt

# Prototypal Inheritance

```
▶ var myMammal = {  
  name : "Herb",  
  get_name: function() {  
    return this.name;  
  },  
  says : function() {  
    return this.saying || "";  
  }  
};
```

# Prototypal Inheritance

- ▶ `var myCat = Object.beget(myMammal);`  
`myCat.name = "Henrietta";`  
`myCat.saying = "meow";`  
...
- ▶ `(.beget())` ist eine Crockford'sche Funktion)

# Prototypal inheritance II - Aggregate

```
▶ var swimFeature = function(that) {  
  that.swim = function() {  
    alert("swim");  
  }  
}
```

```
myDolphin = Object.beget(myMammal);  
myDolphin.saying = "blubb";  
swimFeature(myDolphin);
```

```
▶ // http://trash.chregu.tv/advjs.html
```

# Prototypal inheritance III - Aggregate

```
▶ var swimFeature = {  
  swim : function() {  
    alert("swim");  
  },  
  dive : function() {  
    alert("dive");  
  }  
}  
myDolphin = {};  
YAHOO.lang.augmentObject(myDolphin, myMammal);  
YAHOO.lang.augmentObject(myDolphin, swimFeature);  
myDolphin.saying = "blubb";
```

# jslint

# jslint

- \* Use `[]` and `{}` instead of `new Array()` and `new Object()`
- \* Use `foo.bar` instead of `foo['bar']`
- \* Don't use globals, declare local vars with `var`
- \* Use `function(` for known and `function (` for anonymous
- \* Use `===` especially for comparison to `null`, `false`, `0` and `''`
- \* Write the semicolons!
  
- \* Use it from [jslint.com](http://jslint.com) or javascript tools for TextMate

# Sources

- \* history: <http://javascript.crockford.com/survey.html>
- \* Data Types: <http://javascript.crockford.com/survey.html>
- \* Classical Inheritance: <http://javascript.crockford.com/inheritance.html>
- \* Prototypal Inheritance in JavaScript: <http://javascript.crockford.com/prototypal.html>
- \* Yahoo namespaces: <http://developer.yahoo.com/yui/yahoo/#namespace>
- \* Module Pattern: <http://yuiblog.com/blog/2007/06/12/module-pattern/>
- \* Private Members: <http://www.crockford.com/javascript/private.html>
- \* jslint: <http://jshint.com>
- \* Yahoo.lang.augment/extend: <http://developer.yahoo.com/yui/yahoo/#extend>