

JavaScript, ECMAScript 2015 en TypeScript

CM Telecom – 1 februari 2016



Peter Kassenaar
info@kassenaar.com

Peter Kassenaar

Over Peter Kassenaar:

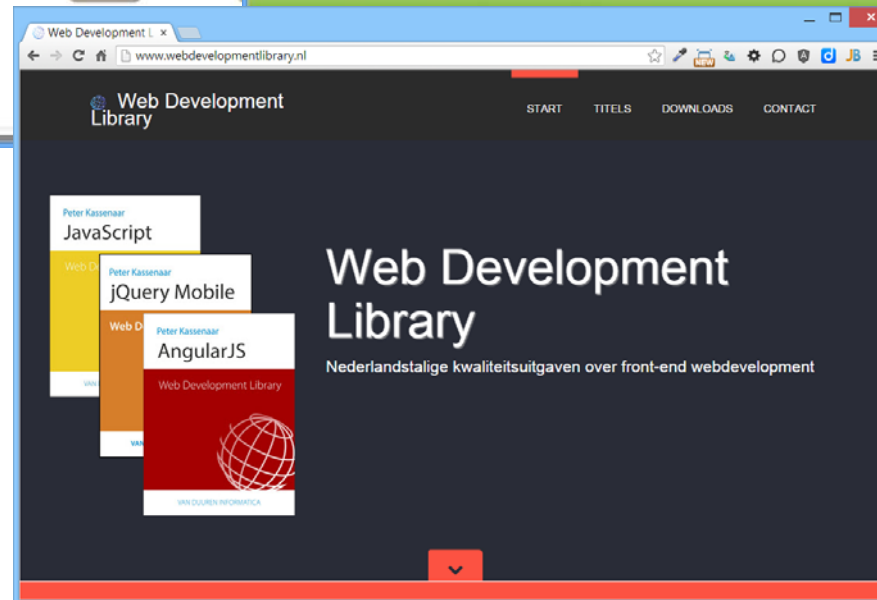
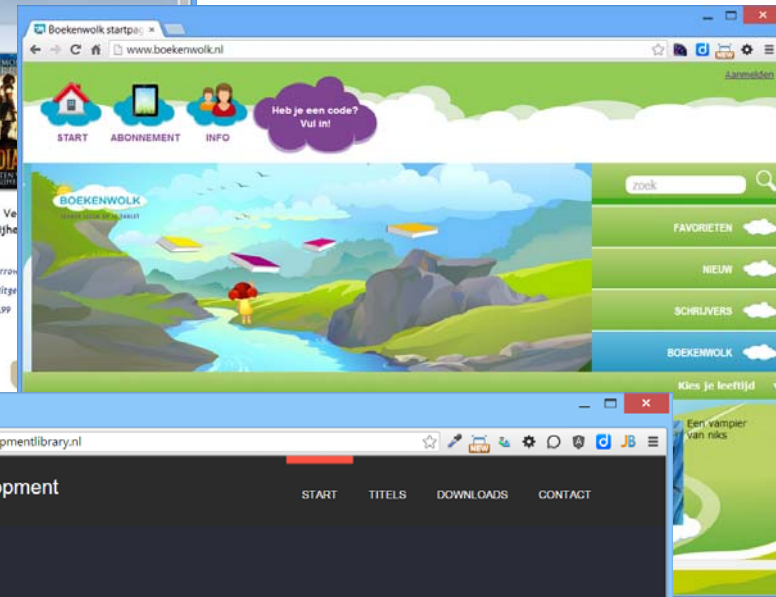
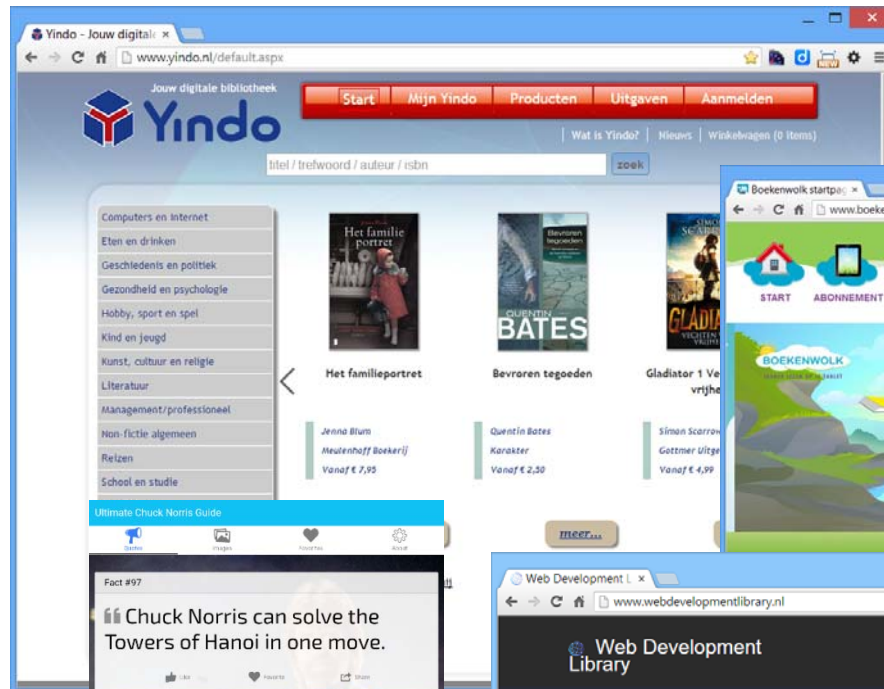
- Trainer, auteur, developer – sinds 1996
- Specialisme: *"Everything JavaScript"*
- JavaScript, ES6, AngularJS, NodeJS, jQuery, PhoneGap, TypeScript

www.kassenaar.com/blog

info@kassenaar.com

Twitter: [@PeterKassenaar](https://twitter.com/PeterKassenaar)





Over jullie...



Voorkennis webdevelopment, (mobile/web-) apps?

Kennis JavaScript, TypeScript?

Voorkennis andere frameworks of (web-)talen?

Verwachtingen van de cursus?

Concrete projecten?

Materialen

Software (downloads)

Handouts (PPTX/PDF)

Oefeningen (papier)

Websites (online)

The screenshot shows the ECMAScript Compatibility Table (ES6 compatibility table) with the following structure:

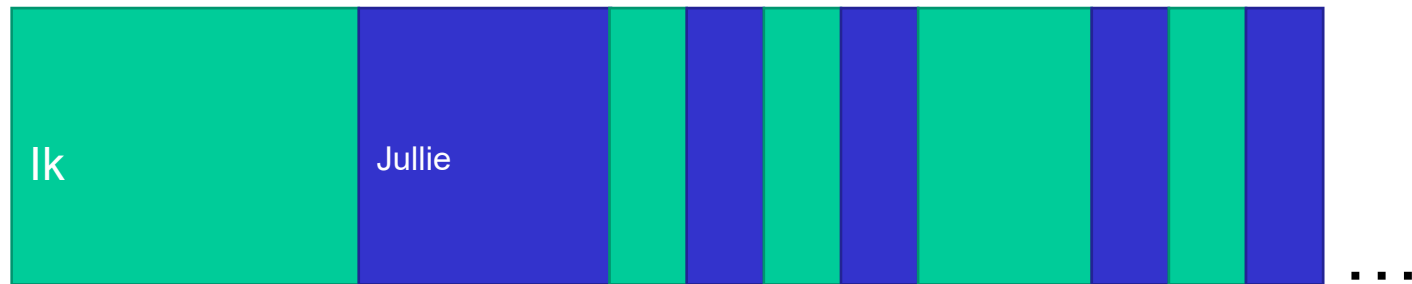
- Header:** ECMAScript 5, 6, 7, Intl, non-standard, compatibility table. By kangax, Grunp, webbedspace, Fork, 236.
- Legend:** V8, SpiderMonkey, JavaScriptCore, Chakra, Carakan, K5, Other. Minor difference (1 point), Useful feature (2 points), Significant feature (4 points), Landmark feature (8 points).
- Table Columns:** Feature name, Current browser, Tracur, Babel + core-js, Closure, JSX, Type-Script + core-js, es6-shim, IE 11, Edge 12, Edge 13, FF 38 ESR, FF 43, CH 48, OP 35, SF 6.1, SF 7.
- Table Rows:**
 - Optimisation:** proper tail calls (tail call optimisation) (0/2, 1/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2).
 - Syntax:**
 - default function parameters (4/7, 0/7, 4/7, 0/7, 4/7, 0/7, 0/7, 0/7, 0/7, 0/7, 3/7, 4/7, 0/7, 0/7, 0/7, 0/7, 0/7).
 - rest parameters (4/5, 4/5, 2/5, 3/5, 3/5, 0/5, 0/5, 5/5, 5/5, 4/5, 5/5, 5/5, 0/5, 0/5, 0/5, 0/5, 0/5).
 - spread (...) operator (15/15, 13/15, 9/15, 2/15, 4/15, 0/15, 0/15, 12/15, 15/15, 15/15, 15/15, 15/15, 0/15, 5/15, 9/15, 0/15, 0/15).
 - object literal extensions (6/6, 6/6, 4/6, 5/6, 6/6, 0/6, 0/6, 6/6, 6/6, 6/6, 6/6, 6/6, 0/6, 1/6, 5/6, 0/6, 0/6).
 - for...of loops (9/9, 8/9, 6/9, 2/9, 3/9, 0/9, 0/9, 6/9, 7/9, 7/9, 7/9, 7/9, 0/9, 2/9, 8/9, 0/9, 0/9).
 - octal and binary literals (2/4, 4/4, 2/4, 0/4, 4/4, 3/4, 0/4, 4/4, 4/4, 4/4, 4/4, 4/4, 0/4, 0/4, 4/4, 0/4, 0/4).
 - template strings (4/5, 4/5, 3/5, 4/5, 3/5, 0/5, 0/5, 4/5, 5/5, 5/5, 5/5, 5/5, 0/5, 0/5, 5/5, 0/5, 0/5).
 - Backtick " and " flags (2/4, 2/4, 0/4, 0/4, 0/4, 0/4, 0/4, 2/4, 4/4, 2/4, 2/4, 0/4, 0/4, 0/4, 0/4, 0/4, 0/4).
 - destructuring declarations (19/22, 21/22, 14/22, 12/22, 14/22, 0/22, 0/22, 0/22, 0/22, 19/22, 19/22, 0/22, 0/22, 0/22, 19/22, 0/22, 0/22).
 - destructuring assignment (22/24, 24/24, 11/24, 11/24, 18/24, 0/24, 0/24, 0/24, 0/24, 20/24, 21/24, 0/24, 0/24, 12/24, 21/24, 0/24, 0/24).
 - destructuring parameters (18/23, 21/23, 14/23, 13/23, 14/23, 0/23, 0/23, 0/23, 0/23, 18/23, 18/23, 0/23, 0/23, 0/23, 18/23, 0/23, 0/23).
 - Unicode code point escapes (1/2, 1/2, 1/2, 0/2, 1/2, 0/2, 0/2, 2/2, 2/2, 0/2, 1/2, 2/2, 0/2, 0/2, 2/2, 0/2, 0/2).
 - new.target (0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 0/2, 1/2, 0/2, 2/2, 0/2, 0/2, 0/2, 0/2, 0/2).
 - Bindings:**
 - const (6/8, 6/8, 6/8, 0/8, 6/8, 0/8, 8/8, 8/8, 8/8, 8/8, 8/8, 5/8, 1/8, 1/8, 1/8, 1/8, 1/8).
 - let (8/10, 8/10, 8/10, 0/10, 7/10, 0/10, 8/10, 8/10, 8/10, 0/10, 0/10, 5/10, 0/10, 0/10, 0/10, 0/10, 0/10).
 - bind-level function declarations (1/2) (Yes, Yes, Yes, Yes, No, No, No, Yes, Yes, Yes, No, No, Yes, No, Yes, No, No).

<https://kangax.github.io/compat-table/es6/>

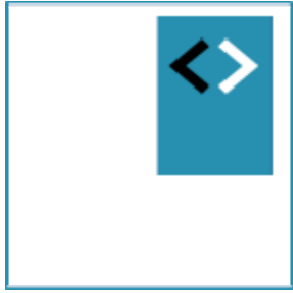
Agenda

- JavaScript / ECMAScript 5 advanced concepts
 - Design patterns
 - Prototypal Inheritance
- ECMAScript 2015
 - New keywords
 - New parameters options
 - Transpiling
- TypeScript
 - Using TypeScript
 - Typing variables and functions
 - Classes
 - Interfaces / Generics

Globale werkwijze



Vragen?



ES5 Advanced concepts

Closures, design patterns, inheritance

Credits : Dan Wahlin



Blog

<http://weblogs.asp.net/dwahlin>



Twitter

@DanWahlin



<http://www.pluralsight.com>

Contents

- *Problem:*
 - Function Spaghetti Code
- **Closures to the Rescue**
- *Possible solutions:*
 - Object Literals and Namespaces
 - 1. Prototype Pattern
 - 2. Revealing Module Pattern
 - 3. Revealing Prototype Pattern

Function Spaghetti Code

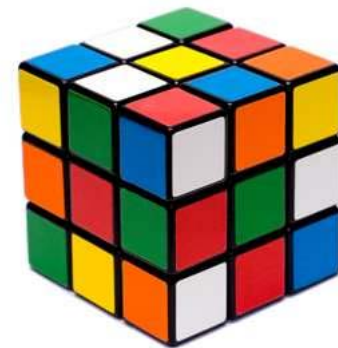


```
<script type="text/javascript">
  window.onload = function () {
    currNumberCtl = document.getElementById('currNumber');
    eqCtl = document.getElementById('eq');
  };
  var eqCtl,
      currNumberCtl,
      operator,
      operatorSet = false,
      equalsPressed = false,
      lastNumber = null;
  function add(x,y) {
    return x + y;
  }
  function subtract(x, y) {
    return x - y;
  }
  function multiply(x, y) {
    return x * y;
  }
  function divide(x, y) {
    if (y == 0) {
      alert("Can't divide by 0");
      return 0;
    }
    return x / y;
  }
  ...

```

Problems with Function Spaghetti Code

- Mixing of concerns
- No clear separation of functionality or purpose
- Variables/functions added into global scope
- Potential for duplicate function names
- Not modular, not distributable
- Not easy to maintain
- No sense of a “container”



Better - Ravioli Code



Advantages of Ravioli Code

- Objects encapsulate and decouple code
- Loosely coupled
- Separation of Concerns
 - Variables/functions are scoped
 - Functionality in *closures*
- Easier to maintain

What is a Closure?

“...an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.”

~ Douglas Crockford

Non-Closure Example


```
function myNonClosure() {  
    var date = new Date();  
    return date.getMilliseconds();  
}
```



Variable lost after
function returns

Closure Example

```
function myClosure() {  
    var date = new Date();  
  
    //nested function  
    return function () {  
        return date.getMilliseconds();  
    };  
}
```



Variable stays around
even after function
returns

Other Example

- Force correct order of count/parameters between async (AJAX) calls

```
49 //*****
50 // 2. Verbeterde poging, nu met closure
51 //*****
52 // Load initial data: de getoonde pagina en de vorige- en volgende pagina. 3 in totaal.
53 var index = -1;
54 BW.pagesHaveLoaded = false;
55
56 // for-lus, ophalen 3 pagina's.
57 for (i = 0; i < 3; i++) {
58     // JavaScript-closure, om te voorkomen dat i een andere waarde heeft als de bitmapstring is
59     // opgehaald door de AJAX-call (en dus de pagina's in verkeerde volgorde in de SwipeView-array
60     // terechtkomen). Credits: http://stackoverflow.com/questions/2405064/ajax-call-in-for-loop-wont-return-values-to-correct-array-positions
61     (function (i) {
62         page = i == 0 ? BW.slides.length - 1 : i - 1;
63         $.ajax({
64             type: 'GET',
65             url: BW.slides[page].src,
66             success: function (yindoPermission) {
67                 el = document.createElement('img');
68                 el.className = 'loading';
69                 el.src = 'data:image/png;base64,' + yindoPermission.pageBitmapString;
70                 el.onload = function () {
71                     this.className = '';
72                     this.className = BW.deviceOrientation;
73                     $('#loadingIndicator').hide();
74                     // <snip>
75                 };
76                 BW.gallery.masterPages[i].appendChild(el); // <== NU gaat het goed, want i is geïsoleerd in de scope van de closure.
77                 index++;
78             }
79         });
80     })(i); // Nieuwe scope creëren door de functie aan te roepen met huidige i als parameter.
81 }
82
83
```

Further reading on closures

MDN is being redesigned! Want to help beta-test it? (Sign up for and) sign in to your MDN account, and then select **Beta user** in your profile.

MOZILLA DEVELOPER NETWORK

READ DOCS • MAKE APPS • BUILD & USE FIREFOX • SEE & SUBMIT DEMOS • GET INVOLVED •

powered by Google

Search MDN

Web technology for developers • JavaScript • JavaScript Guide • Closures

Closures

Read content offline

Closures are functions that refer to independent (free) variables.

In short, variables from the parent function of the closure remain bound from the parent's scope.

Consider the following:

```
1 function init() {
2   var name = "Mozilla"; // name is a local variable created by init
3   function displayName() { // displayName() is the inner function, a closure
4     alert(name); // displayName() uses variable declared in the parent function
5   }
6   displayName();
7 }
8 init();
```

TABLE OF CONTENTS

- Practical closures
- Emulating private methods with closures
- Creating closures in loops: A common mistake
- Performance considerations

TAGS FILES

init() creates a local variable `name` and then a function called `displayName()`. `displayName()` is the inner function (a *closure*) — it is defined inside `init()`, and only available within the body of that function. Unlike `init()`, `displayName()` has no local variables of its own, and instead reuses the variable `name` declared in the parent function.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Closures>

StackExchange • Peter Kassenaar 1 chat meta about help search

stackoverflow Questions Tags Users Badges Unanswered Ask Question

How do JavaScript closures work?

▲ Like the old **Albert Einstein** said: "If you can't explain it to a six-year old, you really don't understand it yourself". Well, I tried to explain JavaScript closures to a 27-year old friend and completely failed.

▼ 2083

☆ 1497

How would you explain it to someone with a knowledge of the concepts which make up closures (e.g. functions, variables and the like), but does not understand closures themselves?

EDIT:

I have seen the **Scheme** example given in Stack Overflow, and it did not help.

javascript closures

share | edit

edited Oct 31 at 8:35

community wiki
10 revs, 6 users 35%
Peter Montesen

102 Closures are a difficult concept to come to grips with on the first try. That's why SO is the perfect way to get this information in a coherent, straightforward series of well written explanations all from different perspectives. I.e. Send the 6-year old here to find the explanation that *they* understand. — **Jess Telford** May 3 '12 at 1:35

2 The title question "How do JavaScript closures work?" could be considered a much more complex question than what we find in the description, which is asking how you would explain (how to use) closures to a 6-year old. Maybe this explains why many of the answers are not really very simple, but nor are they complete or accurate. — **dialbert** Aug 13 '12 at 17:11

11 Practical explanation of closures - jondavidjohn.com/blog/2011/09/... — **jondavidjohn** Sep 26 '12 at 16:31

40 My problem with these and many answers is that they approach it from an abstract, theoretical perspective, rather than starting with explaining simply why closures are necessary in Javascript and the practical situations in which you use them. You end up with a tldr article that you have to slog through, all the time thinking "but, why?". I would simply start with: closures are a neat way of dealing with the following two realities of JavaScript: a. scope is at the function level, not the block level and, b. much of what you do in practice in JavaScript is asynchronous/event driven. — **Jeremy Burton** Mar 8 at 17:22

tagged

javascript 484208

closures 2435

asked 6 years ago

viewed 292948 times

active 14 days ago

CAREERS 2.0

Medior Back end developer (.NET / MVC) in The Hague / Den...

Kamermet
The Hague, Netherlands

ASP.NET Developer
UNIT4
Utrecht, Netherlands

Systemarchitect
Brayzeel Keukebas
Bergen op Zoom, Netherlands

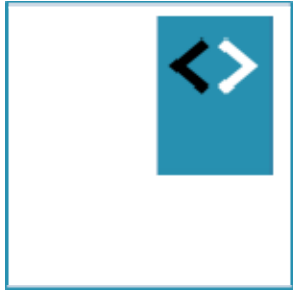
Linked

5 Same JS closure loop issue - but SO's answers aren't working

2 Confused by closures in JavaScript

What is the purpose of

<http://stackoverflow.com/questions/111102/how-do-javascript-closures-work>



Achieving Ravioli Code

Patterns to the rescue

Simplest - Encapsulating Data with Object Literals

- Quick and easy
- All members are available
- Best suited for data

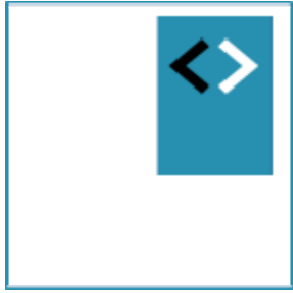
```
var calc = {  
    tempValue: 1,  
    add: function(x, y){  
        return x + y;  
    }  
};
```


Namespaces

- Encapsulate your code in a namespace:

```
var myNS = myNS || {};  
  
myNS.calc = {  
  tempValue: 1,  
  add: function(x, y) {  
    return x + y;  
  }  
};
```

Verdict: quick and simple. Suitable for smaller projects, where no modularity or (re)distribution is required



JavaScript patterns

Prototype pattern, Revealing module pattern, Revealing prototype pattern

1. The Prototype Pattern

- **Benefits:**

- Leverage JavaScript's built-in features
- "Modularize" code into re-useable objects
- Variables/functions taken out of global namespace
- Functions loaded into memory once
- Possible to "override" functions through prototyping



- **Challenges:**

- "this" can be tricky
- Constructor separate from prototype definition



Prototype Pattern Structure

//Constructor defines properties and inits object

```
var Calculator = function (eq) {  
    this.eqCtl = document.getElementById(eq);  
    ...;  
};
```

//Prototype defines functions using JSON syntax

```
Calculator.prototype = {  
    add: function (x, y) {  
        return x + y;  
    },  
    subtract: function (x, y) {  
        return x - y;  
    },  
    ...  
}
```

```
var calc = new Calculator('eqCtl');  
calc.add(2,2);
```

2. The Revealing Module Pattern

- **Benefits:**



- “Modularize” code into re-useable objects
- Variables/functions taken out of global namespace
- Expose only public members

- **Challenges:**



- Functions may be duplicated across objects in memory when not using singleton
- Not easy to extend
- Some complain about debugging

Revealing Module Pattern Structure

```
var calculator = function() {  
    var eqCtl,  
        ...  
  
    init = function(equals, currNumber) {  
        eqCtl = equals;  
        currNumberCtl = currNumber;  
    },  
  
    add = function(x, y) {  
        return x + y;  
    },  
  
    subtract = function(x, y) {  
        return x - y;  
    },  
  
    return {  
        init: init,  
        add: add,  
        ...  
    };  
}(); // <== IIFE patroon
```

calculator.add(2,2);

3. The Revealing Prototype Pattern

- **Benefits:**



- Combines Prototype and Revealing Module patterns
- “Modularize” code into re-useable objects
- Variables/functions taken out of global namespace
- Expose only public members
- Functions loaded into memory once
- Extensible

- **Challenges:**



- "this" keyword can be tricky
- Constructor separate from prototype definition

Revealing Prototype Pattern Structure

```
var Calculator = function (eq) {  
    this.eqCtl = document.getElementById(eq);  
};
```

```
Calculator.prototype = function() {  
    var doAdd = function (x, y) {  
        var val = x + y;  
        this.eqCtl.innerHTML = val;  
    };  
    return {  
        add : doAdd  
    };  
}();
```

```
var calc = new Calculator('eqCtl');  
calc.add(2,2);
```


Summary

- **Function spaghetti code is bad, Ravioli code is good**
- **Closures provide encapsulation**
- **Key patterns:**
 - 1. Prototype Pattern
 - 2. Revealing Module Pattern
 - 3. Revealing Prototype Pattern
- **Which pattern should you use? You decide....**

More info

Sample Code, Book, and Videos

<http://tinyurl.com/StructuringJSCode>



Structuring JavaScript Code

This course walks through several key patterns that can be used to encapsulate and modularize JavaScript code. Throughout the course you'll learn how closures and other techniques can be used to better organize your JavaScript code and make it easier to re-use and maintain in HTML5 applications.



JavaScript Patterns
JumpStart Guide

Clean up your JavaScript Code

Dan Wahlin



Authored by: [Dan Wahlin](#)

Duration: 2h 10m

State

Released: 12/12/2011

Meer over JavaScript design patterns

Addy Osmani: Essential JavaScript design patterns

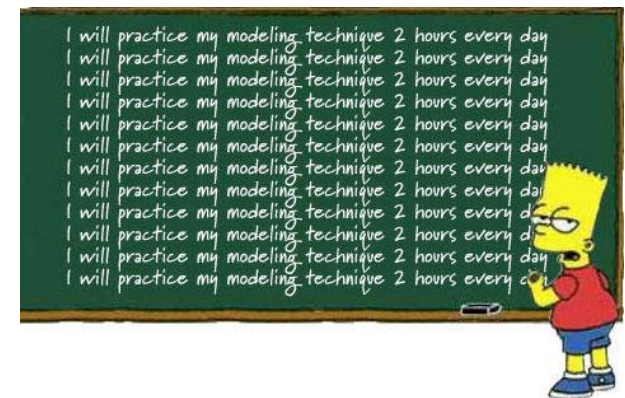
<http://addyosmani.com/resources/essentialjsdesignpatterns/book/>

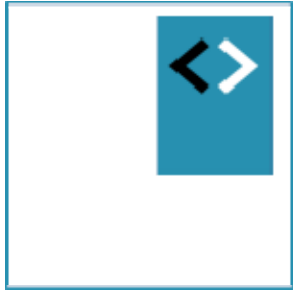
Table Of Contents

- Introduction
- What is a Pattern?
- "Pattern"-ity Testing, Proto-Patterns & The Rule Of Three
- The Structure Of A Design Pattern
- Writing Design Patterns
- Anti-Patterns
- Categories Of Design Pattern
- Summary Table Of Design Pattern Categorization
- JavaScript Design Patterns
 - Constructor Pattern
 - Module Pattern
 - Revealing Module Pattern
 - Singleton Pattern
 - Observer Pattern
 - Mediator Pattern
 - Prototype Pattern
 - Command Pattern
 - Facade Pattern
 - Factory Pattern
 - Mixin Pattern
 - Decorator Pattern
 - Flyweight Pattern
- JavaScript MV* Patterns
 - MVC Pattern
 - MVP Pattern
 - MVVM Pattern

Oefening

- Bekijk de voorbeelden, of maak zelf een object
(Person, Product, Car, ...)
- Breidt dit uit op elk van de drie manieren
 - Prototype pattern
 - Revealing module pattern
 - Revealing prototype patterns
- Maak een herdistribueerbare module van je object en bespreek/leg uit aan collega hoe hij werkt.





JavaScript inheritance

OO met klassiek JavaScript

JavaScript inheritance

Algemeen: maak een ParentClass, of BaseClass

```
// parent class. Convention: classes start with Uppercase
function ParentClass(){
  this.parentProp1 = 'Hello';
  this.parentMethod1 = function (arg1){
    return arg1 + 'parent method 1 return data';
  }
}
```

Maak een afgeleide klasse, of ChildClass

```
//child class
function ChildClass(){
  this.childProp1 = 'Goodbye';
  this.childMethod1 = function (arg1){
    return arg1 + 'Child method 1 return data';
  }
}
```

Zorg er voor dat het prototype van ChildClass verwijst naar ParentClass en maak de instanties

```
// make the child class inherit all parent class characteristics by
// using prototype property.
ChildClass.prototype = new ParentClass();

// instantiate class
var instance1 = new ChildClass();
```

Optioneel : override methods uit de ParentClass

```
// if desirable, you can override the parent methods.
// Again, use prototype
ChildClass.prototype.parentMethod1 = function parentMethod1(arg1){
    return arg1 + 'I have overridden parent method 1';
}
```

YouTube NL

Uploaden

Peter Kas

JAVASCRIPT INHERITANCE TUTORIAL

Object Parent / Child Relationships
aka:
Object Class / Subclass Relationships

adamkhoury.com

00:05 / 11:19

JavaScript Inheritance Tutorial Object Oriented Class Programming

De macht ligt niet bij de bank

Probeer nu gratis

<http://youtu.be/pu08qQCmw8I>

Oefening

- Maak zelf een Parent – Child relatie tussen classes
 - Bijvoorbeeld : voertuigen, personen, producten, etc.
 - Instantieer verschillende instanties en check of je in de child-classes members of methodes uit de parent-class kunt gebruiken.
 - Override een methode uit de parent-class in de child-class en test of dit werkt.

