JAVASCRIPT TRAINING

Case 6 - Uitwerkingen

Download de sources van de uitwerkingen

Opdracht 1 - Inheritance via prototyping

5. Als het goed is zie je twee logregels met 1 maal de gebruikers array als JSON en 1 maal de todos array als JSON gelogged. De prefix is echter nu nog "undefined", hoe kan dit? Waar komt deze prefix vandaan? Los dit probleem op door een aanpassing te maken in de TodoView class, gebruik als prefix "TodoView".

TodoView extend nu wel van View, echter roept de TodoView constructor geen call naar de super constructor (= die van View) uit. Hierdoor wordt de viewNaam niet gezet. De undefined prefix komt dus doordat de logVariabele methode op de view gebruik maakt van de viewNaam om te loggen, een variabele die niet gezet is, geeft als String representatie "undefined" terug op de console.

Opdracht 2 - Scoping

1. Probeer in init.js onderaan nogmaals een constructor function genaamd TodoView te declareren. Refresh vervolgens je browser, wat gebeurd er? Hoe komt dit?

Door nogmaals een constructor function genaamd TodoView te maken, wordt de oude constructor function overschreven. Tevens alle properties (dus ook gezette prototype methodes) worden overschreven. In feite overschrijf je hiermee dus het volledige object, waardoor de eerder gedeclareerde methodes die worden aangeroepen op het object niet meer beschikbaar zijn. Met als gevold "methodeXXX is not a function" meldingen op de error console in FireBug.

2. Haal de net gemaakt constructor function weer weg, refresh je browser en verifieer of de pagina weer werkt. Voer middels FireBug een console.log uit van de variable "todoViewInstance" en van "window.todoViewInstance". Wat valt je op? Welke conclusie kun je hieruit voorzichtig trekken? Voer een === vergelijking tussen de twee uit om dit te verifieren.

De todoViewInstance die werd gedeclareerd, werd in de global scope gedeclareerd. In je browser betekend dit dat alles wat je declareert op het window object terecht komt. Dus een statement als "window.x" en "x" refereren beide naar hetzelfde object "x". Vandaar dat een === vergelijking ook true oplevert.

Alles wat je op top level niveau declareert, komt in de global scope terecht. Alles wat je binnen een function declareert blijft binnen die function als je netjes gebruik maakt van het "var" keyword. Als je dit keyword vergeet, dan wordt alsnog de variabele aan de global scope toegewezen.

3. Herhaal stap 2 met de variabele "gebruikers". Herhaal stap 2 vervolgens met de classes "View" en "TodoView". Wat valt je op?

Ook View en TodoView zijn in de global scope, en dus op het window object binnen je browser, beschikbaar. Ondanks dat je code hebt opgesplitst in meerdere files, betekend nog niet dat iedere file binnen een eigen scope of iets dergelijks uitgevoerd wordt. In je browser heb je 1 javascript context, die bestaat uit alle ingelezen javascript bestanden. Dat betekend in dit geval dus dat View en TodoView, maar ook de variabele todoViewInstance binnen dezelfde global scope "leven" in je browser.

4. Op welke manier kunnen we dit probleem van scoping oplossen, dus hoe zorg je ervoor dat niet alles binnen de global scope zit?

Door gebruik te maken van het module pattern, en vanuit een module alleen die variabele/classes te "exporten" die van buitenaf (vanuit de global scope) beschikbaar mogen zijn. Middels het module pattern voorkom je dat je andere libraries in de weg zit, of dat andere libraries jouw code in de weg zitten. Middels het module pattern scope je je classes en variabelen tot je eigen gedefinieerde module. Andere modules kunnen op deze manier dezeflde class/variabele namen gebruiken, zonder dat die in de global scope komen en mogelijk andere variabele/classes overschrijven.

Het module pattern kun je vergelijke met het namespacen van je code.

PUBLISHED ON March 6th, 2014