Efficiënt scripten met

jQuery 1.11

Ontwerpen van interactieve websites met HTML5, CSS3 en jQuery

Patrick Verhaert

Efficiënt scripten met jQuery 1.11 (jQuery 2.1)

Ontwerpen van interactieve websites met HTML5, CSS3 en jQuery

Patrick Verhaert

This book is for sale at http://leanpub.com/jq11

This version was published on 2015-12-15

Tweet This Book!

Please help Patrick Verhaert by spreading the word about this book on Twitter!

The suggested hashtag for this book is #jqueryBoek.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#jqueryBoek

Inhoudsopgave

1. Elementen selecteren.

Een jQuery statement bestaat steeds uit twee of meer delen. Het eerste deel is altijd een selector. De volgende delen beschrijven de acties die u op de selector uitvoert.

Een voorbeeld:

```
$('p.foo').addClass('bar').show('slow');
```

- **`$('p.foo')`**: selecteer alle p-tags met het class-attribuut *.foo*.
- **`.addClass('bar')`**: voeg aan deze selectie de class *.bar* toe.
- **`.show('slow');`** en maak de selectie zichtbaar.

Een goede selectie maken is bijgevolg cruciaal om een krachtig script uit te voeren.

JavaScript kent een zeer beperkte set aan selectors.

U kan bijvoorbeeld een bepaald id (`document.getElementById(id)`) of een bepaalde tag opvragen (`document.getElementByTagName(tag)`), maar wat als u de derde rij in een tabel wilt ophalen of alle links met een externe URL? Dit is onmogelijk met één commando.

Dit hoofdstuk beschrijft de belangrijkste selectiemethodes binnen jQuery. De syntax is eenvoudig. Deze volgt immers de selectiemethodes van CSS3, aangevuld met enkele jQuery-specifieke selectiemethodes.

De voorbeelden die volgen, kan u uittesten op het oefenbestand **selectorTest.html**.

⋆ Open **selectors/selectorTest.html**.

⋆ De voorbeelden die volgen, kan u uittesten op dit oefenbestand.

Geef in het tekstveld een selectie in (bv: *p#paragraaf1*). Het script op de pagina gaat op zoek naar de selectie en voegt aan de selectie de class *.highLight* toe. Deze class tekent een groen kader rond de selector, maakt de achtergrond geel en kleurt de tekst rood.

Een overzicht van de verschillende selectors vindt u hier:
http://api.jquery.com/category/selectors/basic-css-selectors/

1.1 Basis selectors.

Zie: http://api.jquery.com/category/selectors/basic-css-selectors/

1.1.1 Tags, classes en id's.

Volgende voorbeelden gebruikt u waarschijnlijk dagelijks in CSS:

```
$('a')              // selecteer alle a-tags
$('a, tr')          // selecteer alle a-tags en alle tr-tags
$('#paragraaf1')    // selecteer de tag met id #paragraaf1
$('.sublist')       // selecteer alle tags met class .sublist
```

1.1.2 Universele selector.

Het sterretje is, net zoals in CSS, de universele selector voor alle tags.

```
$('*')              // selecteer alle tags
```

1.2 Hiërarchische selectors.

Zie: http://api.jquery.com/category/selectors/hierarchy-selectors/

Hiërarchische selectors kan u best vergelijken met een stamboom. Bekijk in onderstaande code hoe de tags in elkaar genest zijn.

```
<table class="layoutTabel">
    <thead>
        <tr>
            <th>Opleiding:</th>
            <th>Datum:</th>
            <th>Lokaal:</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Dreamweaver</td>
            <td>15 december</td>
            <td>PC1</td>
        </tr>
    </tbody>
</table>
```

De hiërarchische selectors zijn:

Parent en child

Parent en child zijn geneste tags op het eerste niveau.
De thead-tag en de tbody-tag staan rechtstreeks onder de table-tag en gedragen zich in een *parent-child* relatie.
table is een **parent** van *thead* of
thead is een **child** van *table*.

Siblings

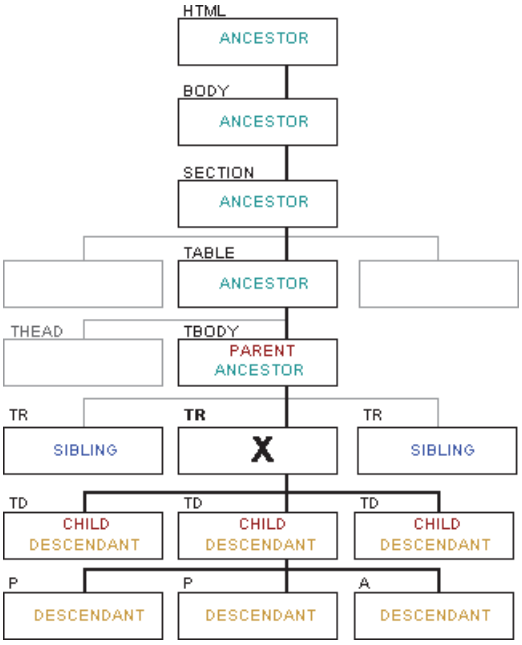
Siblings (broers of zussen) zijn elementen die op gelijk niveau staan.
De drie td-tags binnen een tr-tag zijn **siblings** van elkaar.

Ancestor (voorouder)

Een *ancestor* kan de *parent* zijn, maar ook de *parent* van de *parent*, of nog een verdere relatie.
th heeft als **parent** *tr*.
th heeft als **ancestor** *tr*, maar ook *thead*, *table* en *body*.

Descendant (nakomeling)

Een *descendant* kan een *child* zijn, maar ook een *child* van een *child* of nog een verdere relatie.
table heeft als **child** *thead* en *tbody*.
table heeft als **descendant** *thead*, *tbody*, maar ook *tr*, *th* en *td*.



⋆ Controleer onderstaande selecties op het oefenbestand.
Enkel het deel tussen de aanhalingstekens invullen.
Bijvoorbeeld: `table tr` en niet `$('table tr')`.

1.2.1 Descendant selector: $('ancestor descendant').

```
$('table tr')    // alle tr-tags binnen een tabel
$('table td')    // alle td-tags binnen een tabel
$('table td a')  // enkel de links  in td-tags van een tabel
$('ul *')        // alle tags binnen een ul-tag
```

1.2.2 Child selector: $('parent > child').

```
$('table > tbody')       // de tbody-tags binnen een tabel
$('table > td')          // de td-tags binnen een tabel (*)
$('table > tbody > tr')  // alle rijen binnen de tbody-tag
```

(*) Merk op dat in ons voorbeeld geen enkele td-tag rechtstreeks onder de table-tag staat. Er wordt dus niets geselecteerd!

1.2.3 Sibling selector (alle volgende) : $('prev ~ sibling').

```
$('thead ~ tbody')   // alle tbody-tags die volgen op een thead-tag
$('li ~ li')         // alle li-tags die volgen op een li-tag
                     // de eerste li-tag van elke lijst wordt
                     // niet geselecteerd!
$('h3 ~ p')          // alle p-tags die volgen op een h3-tag
```

1.2.4 Sibling selector (dadelijk aangrenzende): $('prev + next').

```
$('thead + tbody')   // alle tbody-tags die DADELIJK volgen op een thead-tag.
$('li + li')         // alle li-tags die DADELIJK volgen op een li-tag
                     // de eerste li-tag van een lijst wordt niet geselecteer\
d!
$('h3 + p')          // alle p-tags die DADELIJK volgen op een h3-tag
```

In het laatste voorbeeld worden *paragraaf2* en *paragraaf3* niet geselecteerd. Tussen *paragraaf 1* en *paragraaf 2* staat immers nog een lijst.

1.3 Basisfilters.

Zie : http://api.jquery.com/category/selectors/basic-filter-selectors/

De selectors die we tot hiertoe hebben besproken, selecteren telkens een volledig element. `$('tr')` selecteert bijvoorbeeld alle tr-tags in een tabel.

Stel dat we nu enkel de even of oneven rijen in een tabel wensen te selecteren, dan doen we dit aan de hand van filters. Een filter is als het ware een subselectie binnen een hoofdselectie. Alle filters worden met een dubbelpunt als prefix aangeduid. Voor de oneven rijen (rijen met index *1*, *3*, ...) in een tabel wordt dit: `$('tr:odd')`.

Bij sommige filters moet u eveneens een index meegeven. Het eerste element heeft de index *0*. Om bijvoorbeeld de **derde rij** binnen een tabel te selecteren, geeft u als **index de waarde 2** mee `$('tr:eq(2)')`.

1.3.1 Subselecties binnen een reeks.

```
$('table tr:even')   // alle even tr-tags binnen een tabel
$('table tr:odd')    // alle oneven tr-tags binnen een tabel
$('img:first')       // de eerste afbeelding op de pagina
$('img:last')        // de laatste afbeelding op de pagina

$('img:eq(2)')       // de DERDE (niet tweede) afbeelding op de pagina
$('img:lt(2)')       // alle afbeeldingen voor de derde afbeelding
$('img:gt(2)')       // alle afbeeldingen na de derde afbeelding
```

1.3.2 Inverse subselecties binnen een reeks met :not().

```
$('img:not(:eq(2))')  // alle afbeeldingen, behalve de derde afbeelding
```

1.3.3 Overige subselecties.

(De drie laatste selectors kan u niet op het oefenbestand testen.)

```
$('*:header')        // alle hx-tags (h1 t/m h6)
$(':header')         // alle hx-tags (h1 t/m h6)
$('*:focus')         // het element dat op dit moment geselecteerd is
$(':focus')          // het element dat op dit moment geselecteerd is
$('div:animated')    // alle div-tags die op dit moment bewegen
```

1.4 Inhoudsfilters.

Zie : http://api.jquery.com/category/selectors/content-filter-selector/

Deze filters testen op de inhoud (*tekst* of *html*) binnen een bepaald element.

1.4.1 Het element bevat een bepaalde tekst :contains().

```
$('td:contains("PC")')          // alle cellen die de tekenreeks PC bevatten
$('td:contains(PC)')
$('td:not(:contains("PC"))')    // alle cellen die de tekenreeks PC
                                // niet bevatten
$('td:not(:contains(PC))')
```

De tekst is hoofdlettergevoelig, maar de aanhalingstekens zijn niet verplicht.
`contains("PC")` is **NIET** hetzelfde als `contains("pc")`.
`contains("PC")` is **WEL** hetzelfde als `contains(PC)`.

1.4.2 Het element bevat een bepaald element :has().

```
$('td:has(a)')                  // alle cellen die een link bevatten
$('td:not(:has(a))')            // alle cellen die geen link bevatten
```

1.4.3 Lege (:empty) of niet lege (:parent) elementen.

```
$('td:empty')                   // alle lege cellen binnen een tabel
$('td:not(:empty)')             // alle cellen die niet leeg zijn
$('td:parent')                  // alle cellen die minstens een child hebben
```

1.5 Attribuut selectors.

Vanuit CSS kan u elementen selecteren op basis van hun attribuut. Ook deze eigenschap werd door jQuery overgenomen. Bekijk de broncode van het oefenbestand. Neem bijvoorbeeld een afbeelding. Alle afbeeldingen hebben natuurlijk een src-attribuut en een alt-attribuut. Sommige afbeeldingen hebben ook het title-attribuut.

Zie : http://api.jquery.com/category/selectors/attribute-selectors/

1.5.1 Attribuut bestaat `[name]`.

```
$('img[src]')           // alle afbeeldingen met het src-attribuut
$('img[title]')         // alle afbeeldingen met het title-attribuut
$('a[target]')          // alle links met het target-attribuut
$('a:not([target])')    // alle links zonder target-attribuut
```

We kunnen een selectie nog verder verfijnen door de waarde of inhoud van het attribuut te controleren.

1.5.2 Attribuut is gelijk aan `[name="value"]`.

```
$('a[target="_blank"]')     // alle links met target="_blank"
                            // of zonder opgegeven target
```

1.5.3 Attribuut is niet gelijk aan `[name!="value"]`.

```
$('a[target!="_blank"]')    // alle links zonder target
                            // of mét een ander target dan "_blank"
```

1.5.4 Attribuut begint met `[name^="value"]`.

```
$('a[href^="http://"]')     // alle externe html-links
```

1.5.5 Attribuut eindigt op `[name$="value"]`.

```
$('a[href$=".com"]')        // alle links die eindigen op .com
```

1.5.6 Attribuut bevat `[name*="value"]`.

```
$('a[href*="wiki"]')        // alle links die de tekenreeks "wiki" bevatten
```

1.5.7 Attribuut bevat meerdere filters `[filter1][filter2]`.

```
$('a[href^="http://"][title]')   // alle externe links met een title-attribuut
```

1.6 Childfilters.

Een childfilter selecteert het n-de element binnen het parent element.
Zie : http://api.jquery.com/category/selectors/child-filter-selectors/

1.6.1 Eerste child element :first-child.

```
$('p strong:first-child')    // de eerste strong-tag in ELKE paragraaf
$('p strong:first')          // de eerste strong-tag in de EERSTE paragraaf
                             // die een strong-tag heeft
```

1.6.2 Laatste child element :last-child.

```
$('p strong:last-child')    // de laatste strong-tag in ELKE paragraaf
$('p strong:last')          // de laatste strong-tag in de LAATSTE paragraaf
                            // die een strong-tag heeft
```

1.6.3 Eerste sibling element binnen dezefde parent :first-of-type.

```
$('a:first-of-type')        // de eerste a-tag binnen dezelfde parent
```

1.6.4 Laatste sibling element binnen dezefde parent :last-of-type.

```
$('a:last-of-type')         //de laatste a-tag binnen dezelfde parent
```

1.6.5 Het n-de child elementen.

De index start hier vanaf ***1*** en niet vanaf ***0***!
De letter **n** staat voor een geheel getal (0, 1, 2, 3, ...).

```
$('p strong:nth-child(2)')       // de tweede strong-tag
$('p strong:nth-child(3n)')      // strong-tags 3, 6, 9, ...
$('p strong:nth-child(even)')    // alle even strong-tags
$('p strong:nth-child(2n)')      // alle even strong-tags
$('p strong:nth-child(odd)')     // alle oneven strong-tags
$('p strong:nth-child(2n+1)')    // alle oneven strong-tags
$('p strong:nth-last-child(2)') // de voorlaatste strong-tag
$('li:nth-of-type(even)')        // alle even li-tag binnen dezelfde parent
$('a:nth-last-of-type(2)')       // de voorlaatste a-tag binnen dezelfde pare\
nt
```

1.6.6 Het enige child element :only-child().

```
$('p strong:only-child()')   // de strong-tag, maar ENKEL indien dit HET ENIGE
                             // child element binnen de p-tag is
```

1.6.7 Het enige child element van dat type binnen dezefde parent :only-of-type().

```
$('strong:only-of-type()')   // de strong-tag, maar ENKEL indien dit DE ENIGE
                             // strong-tag is binnen dezelfde parent
```

1.7 Formulierfilters.

Zie : http://api.jquery.com/category/selectors/form-selectors/

Filtert formulierelementen van een bepaald type of met een bepaald attribuut.

1.7.1 De type-selector (button, submit, reset, checkbox, ...).

```
$('input')                   // ALLE input-tags.
$('input[type="button"]')    // alle input-tags met type="button"
$('input[type="submit"]')    // alle input-tags met type="submit"
$('input[type="reset"]')     // alle input-tags met type="reset"
```

1.7.2 Overige attributen (:checked, :disabled, :enabled en :selected).

```
$(':checked')    // checkboxen en radiobuttons met een checked-attribuut
                 // ':checked' werkt NIET op een keuzelijst!
$(':disabled')   // alle elementen met een disabled-attribuut
$(':enabled')    // alle elementen die niet disabled staan
$(':selected')   // het gekozen item uit een keuzelijst
                 // ':selected' werkt NIET op checkboxen en radiobuttons!
```

Met `:checked` en `:selected` selecteert u enkel het element, maar haalt u de gekozen waarde nog niet op. De methodes om deze waardes uit te lezen, komen later aan bod.

1.8 Zichtbaarheidsfilters.

Zie : http://api.jquery.com/category/selectors/visibility-filter-selectors/

```
$('p:visible')      // alle zichtbare p-tags
$('p:hidden')       // alle onzichtbare p-tags
```

1.9 Toepassing 1: gemeentelijst filteren (basisversie).

★ Open **selectors/toep_gemeentelijst_1.html**.

De pagina bevat een lijst van alle Vlaamse gemeenten.
Het zoekveld bovenaan de pagina filtert de lijst met gemeentenamen.

```
<body>
...
<input type="text" name="filter" id="filter" ... ">
...
<ul>
    <li><a ... >Aalst</a></li>
    <li><a ... >Aalter</a></li>
    <li><a ... >Aarschot</a></li>
    <li><a ... >Aartselaar</a></li>
</ul>
...
</body>
```

★ Pas het script als volgt aan.

```
1 $(function() {
2     $('#filter').keyup(function(){
3         var filter = $(this).val();
4         $('li').hide();
5         $('li:contains(' + filter + ')').show();
6     });
7 });
```

Telkens u in het tekstveld een letter intypt, wordt het event `keyup()` geactiveerd. De waarde uit het tekstveld komt in de variabele *filter* terecht.
Indien u bijvoorbeeld *ka* invult, wordt getest of dit woord binnen een li-tag voorkomt. De selectoren op lijn 4 en 5 bepalen of de li-tag zichtbaar of onzichtbaar wordt.

Merk op dat op lijn 5 de variabele dynamisch in de selector wordt verwerkt. Voor het zoekwoord *ka* wordt dit: `$('li:contains(ka)').show();`

> Zoeken op ***Ka*** of op ***ka*** een toont een totaal verschillend resultaat. De zoekfunctie is namelijk hoofdlettergevoelig. Met de selecties uit dit hoofdstuk kunnen we dit nog niet oplossen. In het volgend hoofdstuk gaan we dit probleem wegwerken door de selecties verder te verfijnen.

1.10 Toepassing 2: openingsuren markeren.

⋆ Open **selectors/toep_openingsuren.html**.

Openingsuren stadhuis Geel

Geef in elke tabel de rij van vandaag een opvallende achtergrondkleur.
Zie voorbeeld: http://www.openingsuren.com/detail.php?edit=5031593

Van september t.e.m. juni

Dag	Voormiddag	Namiddag	Avond
Zondag	Gesloten	Gesloten	Gesloten
Maandag	9u - 12u	13.30u - 16u	Gesloten
Dinsdag	9u - 12u	13.30u - 16u	17.30u - 19.30u
Woensdag	9u - 12u	13.30u - 16u	Gesloten
Donderdag	9u - 12u	13.30u - 16u	Gesloten
Vrijdag	9u - 12u	Gesloten	Gesloten
Zaterdag	Gesloten	Gesloten	Gesloten

In juli en augustus

Dag	Voormiddag	Namiddag	Avond
Zondag	Gesloten	Gesloten	Gesloten
Maandag	9u - 12u	Gesloten	Gesloten
Dinsdag	9u - 12u	Gesloten	17.30u - 19.30u
Woensdag	9u - 12u	Gesloten	Gesloten
Donderdag	9u - 12u	Gesloten	Gesloten
Vrijdag	9u - 12u	Gesloten	Gesloten
Zaterdag	Gesloten	Gesloten	Gesloten

De pagina bevat twee tabellen waarop we de openingsuren van vandaag markeren. Elke weekdag komt overeen met één rij binnen de tbody-tag. Merk op dat we de tabel ook starten met zondag. Dit is een bewuste keuze omdat we het rijnummer dan makkelijk kunnen koppelen aan de weekdag van het Date-object.

```
<table class="layoutTabel">
    <thead>...</thead>
    <tbody>
        <tr>
            <td><strong>Zondag</strong></td>
            <td>Gesloten</td>
            <td>Gesloten</td>
            <td>Gesloten</td>
        </tr>
        ...
    </tbody>
</table>
```

★ Voeg volgende code toe.

```
1  $(function() {
2      var vandaag = new Date();
3      var dag = vandaag.getDay();
4      $('tbody tr:nth-child(' + (dag + 1) + ')').addClass('nu');
5  });
```

Op lijn 3 halen we de weekdag op. (zondag = 0, maandag = 1, ...). Vervolgens gaan we op lijn 4 dit getal met één verhogen om zo de juiste rij binnen tbody te markeren met de class *.nu*.

> Merk op dat de pagina twee tabellen bevat en dat we daarom `tr:nth-child()` gebruiken en niet `tr:eq()`.
> `tr:eq()` zou enkel een rij in de eerste tabel markeren.

2. Inleiding tot AJAX.

Alle moderne websites maken tegenwoordig gebruik van *AJAX*. Denk maar aan *Gmail*, *Google Drive*, *Facebook* en *Twitter*.

In tegenstelling tot een klassieke webpagina worden bij AJAX-gestuurde pagina's delen van de pagina ververst zonder de volledige webpagina opnieuw in te laden. De pagina's worden hierdoor veel interactiever en intuïtiever. De gebruiker krijgt meer het gevoel dat hij in een desktopapplicatie werkt dan op een website. Websites die hoofdzakelijk gebruik maken van AJAX noemt men ook wel **R**ich **I**nternet **A**pplicaties of **RIA**'s.

AJAX is geen technologie op zich, maar een algemene term voor het ontwerpen van interactieve pagina's waarbij gegevens uit een extern bestand worden opgehaald en vervolgens dynamisch worden getoond. Het grote voordeel van AJAX t.o.v. uitsluitend klassieke webpagina's is dat alle gegevens dynamisch in de browser worden geladen. Alle interactiviteit (sorteren, filteren, …) gebeurt dus volledig binnen de browser zonder dat er een nieuwe connectie met de server nodig is.

De communicatie binnen een klassieke website gebeurt als volgt:

1. De browser laadt een pagina.
2. De gebruiker klikt op een link.
3. De browser vraagt **een volledig nieuwe pagina op**.
4. De webserver stuurt de nieuwe pagina naar de browser.
5. De browser toont de nieuwe pagina.

De communicatie met een AJAX-pagina verloopt als volgt:

1. De browser laadt een pagina.
2. De gebruiker klikt op een link.
3. De JavaScript engine verwerkt de aanvraag en stuurt de aanvraag in de achtergrond door naar een externe pagina.
4. De JavaScript engine **ontvangt de gegevens en past het DOM aan**.
5. De browser toont de nieuwe gegevens en hoeft daarvoor niet de volledige pagina te herladen.

Omdat de browser minder met de webserver communiceert en omdat de verwerking volledig lokaal gebeurt, krijgen we een zeer snelle respons en een snelle pagina update.

In dit hoofdstuk geven we een korte inleiding tot AJAX en verduidelijken we enkele begrippen. De oefeningen komen in de twee volgende hoofdstukken aan bod.

Een volledig overzicht van alle AJAX-methodes vindt u hier:
http://api.jquery.com/category/ajax/

2.1 Historiek.

De techniek om asynchroon gegevens op te halen, bestaat al meer dan 20 jaar. In 1998 ontwikkelde Microsoft een systeem om via een *ActiveX control* gegevens in de achtergrond op te halen. Enkele jaren later implementeerden alle andere browsers een gelijkaardig principe, maar nu gebaseerd op het gestandaardiseerde *XMLHttpRequest protocol* (kortweg *XHR*).

Google was het eerste bedrijf dat *XHR* op grote schaal in zijn toepassingen ging verwerken. In de beginjaren kwamen de externe gegevens uitsluitend uit een XML-document. Zo is de term AJAX ontstaan. **AJAX** was het acroniem voor **A**synchronous **J**avaScript **A**nd **X**ML.

Ondertussen is de omschrijving van AJAX al achterhaald. Via AJAX kunnen we niet enkel XML importeren, maar eveneens tekst, HTML, JSON en JavaScript.

Het is niet zo evident om op een universele manier AJAX te verwerken. De ene browser gebruikt XHR, de andere ActiveX. Gelukkig hoeven wij ons hier geen zorgen over te maken. jQuery zorgt immers voor een correcte verwerking in de verschillende browsers.

2.2 Wat is XML?

XML is het acroniem voor e**X**tensible **M**arkup **L**anguage. Net zoals HTML, beschrijft XML de datastructuur van gegevens, niet de opmaak. Een XML-document is, net zoals HTML, opgebouwd met tags (*nodes* in het XML jargon) en attributen.

Neem als voorbeeld een adresboek in de vorm van een XML-document.

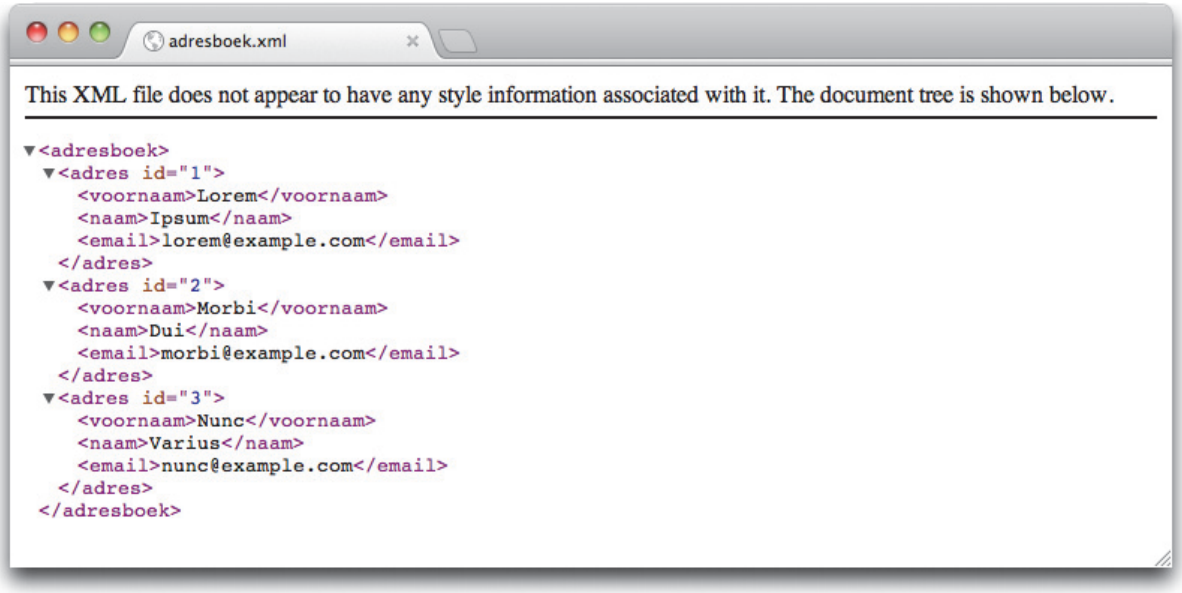
XML begint steeds met de header of proloog. De proloog bevat informatie over de document encoding en de XML versie. Dit is ondermeer belangrijk voor het programma dat de XML-code gaat verwerken (de XML-parser). Na de proloog volgt de rootnode *adresboek*.
Ons adresboek bevat twee personen. Elke persoon staat beschreven in een eigen adresnode met als attribuut een uniek *id*. Binnen de adresnode komen de childnodes: *voornaam*, *naam* en *email*.

```
<?xml version="1.0" encoding="UTF-8"?>
<adresboek>
    <adres id="1">
        <voornaam>Lorem</voornaam>
        <naam>Ipsum</naam>
        <email>lorem@example.com</email>
    </adres>
    <adres id="2">
        <voornaam>Morbi</voornaam>
        <naam>Dui</naam>
        <email>morbi@example.com</email>
    </adres>
    <adres id="3"> ... </adres>
</adresboek>
```

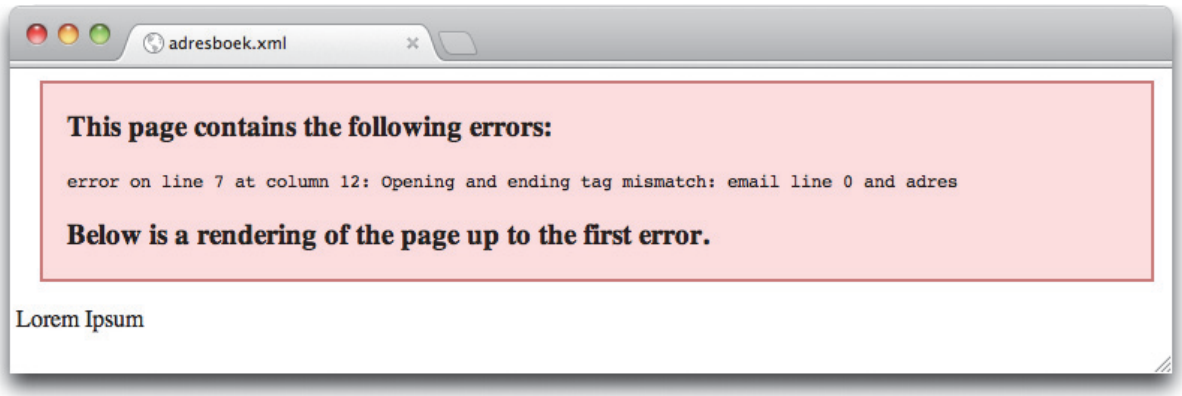
Een goed gestructureerd XML-document is zelfbeschrijvend. Dit wil zeggen dat nodenamen iets vertellen over de inhoud van de node.

Als het document voldoet aan alle syntaxregels van XML, noemt men dit **well-formed** (of goed gevormd). Een well-formed document kan door de meeste parsers, zoals een webbrowser, correct verwerkt worden.

Een well-formed document kan u makkelijk in Firefox of in Chrome testen. Open het XML-document in de browsers. Als de boomstructuur verschijnt, is het document well-formed en zijn de gegevens te verwerken via jQuery.



Indien het document fouten bevat, is het niet well-formed en krijgt u dit te zien:



In een XML-document dat uitsluitend voor eigen gebruik is ontworpen, kan u de nodenamen vrij kiezen. In een universeel, gestandaardiseerd XML-document zoals RSS, ATOM en XHTML ligt de naamgeving vast.

Om de correctheid van het document te controleren, maakt men gebruik van een **DTD** of van een **XML Schema**.

Een DTD of XML schema documenteert als het ware het XML-bestand. Hierin wordt ondermeer

beschreven welke nodes in het XML-document moeten/mogen voorkomen en welke inhoud de nodes bevatten (tekst, enkel getallen, …). Aan de hand van dit controlebestand kan de parser de XML-nodes zowel op syntax als op inhoud valideren.

Een well-formed XML-bestand dat ook nog voldoet aan de bijbehorende DTD of Schema, noemt men een **valid XML-bestand**.

2.3 Wat is JSON?

JSON is het acroniem voor **J**ava**S**cript **O**bject **N**otation. JSON beschrijft, net zoals XML, de datastructuur van gegevens. JSON is een onderdeel van JavaScript, en is bijgevolg relatief eenvoudig in het DOM te verwerken.

De eenvoud van JSON heeft geleid tot een grote populariteit ervan, met name als een alternatief voor XML. Binnen JSON staan gegevens gestructureerd in de vorm van een JavaScriptobject of als een JavaScript array.

Ziehier een JSON equivalent van ons XML adresboek.

```
{
    "adresboek":[
        {
            "id":1,
            "voornaam":"Lorem",
            "naam":"Ipsum",
            "email":"lorem@example.com"
        },
        {
            "id":2,
            "voornaam":"Morbi",
            "naam":"Dui",
            "email":"morbi@example.com"
        },
        {
            ...
        }
    ]
}
```

> Vergelijk de syntax van bovenstaand JSON object met de syntax van een object literal in paragraaf 2.11.4.
> In JSON moeten alle namen en stringwaardes tussen dubbele aanhalingstekens staan, zoniet kan jQuery het bestand niet verwerken!

- Goed: “voornaam”:”Lorem”
- **Fout**: ***voornaam*** : “Lorem”
- **Fout**: ***‘voornaam’*** : “Lorem”
- **Fout**: ***‘voornaam’*** : ***‘Lorem’***

2.4 Requests filteren met GET en POST.

Het is perfect mogelijk om gegevens (HTML, JSON, XML, ...) te verwerken uit een volledig statisch document. Het wordt natuurlijk nog interessanter indien we de gegevens dynamisch vanuit een database kunnen genereren. Vanuit een statische pagina is de response altijd hetzelfde. Op een dynamische pagina is de response afhankelijk van een filter of parameter. Het filteren gebeurt meestal vanuit een formulier.

Neem bijvoorbeeld de zoekfunctie van Google. De response is afhankelijk van de zoekterm die u in het formulier invult.

Zoals u weet, kan u een formulier verzenden via *GET* of via *POST*.

2.4.1 Formulier verzenden met de GET-methode.

GET plakt alle formuliervelden achter de URL. Alle zoekmachines gebruiken GET.

Zoek via Google op **jquery tutorials** en bekijk de URL.

```
http://www.google.be/search?...&q=jquery+tutorials&.....
```

De structuur van de URL met een GET-request is als volgt:
`action?naam1=waarde1&naam2=waarde2&naam3=waarde3`

Action is de URL van de pagina die het formulier zal verwerken. Van elk formulierveld wordt zowel de naam van het veld als de waarde in het veld aan de URL toegevoegd. Alle velden zijn gescheiden door een &-teken.

Omdat GET de waarde of de inhoud van elk object zichtbaar maakt in de URL, kan u deze methode bijvoorbeeld niet gebruiken op een loginpagina. Het paswoord mag immers niet zichtbaar zijn in de URL. De lengte van de URL is ook beperkt tot enkele honderden karakters (browserafhankelijk).

2.4.2 Formulier verzenden met de POST-methode.

De tweede methode, POST, geeft de gegevens via de header door. De gegevens zijn nu niet zichtbaar in de URL. Het aantal karakters dat kan worden doorgestuurd, is vrijwel onbeperkt.

> Voor een AJAX request naar een statische pagina hebt u geen webserver nodig.
> Voor een AJAX request met GET- en POST-variabelen in combinatie met een database hebt u altijd een webserver nodig. Formuliergegevens kan u dan enkel verwerken via een server-side scripttaal zoals PHP of ASP.NET. Hierdoor bent u wel verplicht om via een webserver te werken.

2.4.3 GET-methode zonder formulier.

Omdat de GET-methode alle parameters achter de URL plaatst, is het perfect mogelijk om de parameters dadelijk in een link te verwerken. Dit wordt vaak gebruikt in een master/detail relatie.

De masterpagina toont een beknopt overzicht van alle items. Elk item heeft een link naar de detailpagina en stuurt in de URL een unieke identificatiecode mee.

Neem als voorbeeld de startpagina van Campinia Media[1].

Elke link "*Lees verder*" verwijst naar dezelfde detailpagina:
`http://www.campiniamedia.be/fondslijst_detail.asp?ISBN=xxxxxx`

[1] http://www.campiniamedia.be

De parameter *ISBN* filtert de juiste gegevens uit de database en toont de gedetailleerde informatie over het gevraagde boek.

2.5 Zes soorten AJAX requests.

De methode `$.ajax()` is een jQuery's low-level AJAX implementatie. Dit is tevens de meest uitgebreide, maar ook moeilijkste methode.

Gelukkig zijn er ook nog vijf afgeleide methodes met minder toeters en bellen, maar wel veel eenvoudiger te begrijpen en te gebruiken. Onderstaande tabel geeft een overzicht van de verschillende methodes met hun mogelijkheden en beperkingen.

	Moeilijkheidsgraad	GET request	POST request	Response				
				Tekst	HTML	Script	JSON(P)	XML
$(selector).**load**(url [, data][, callback])	*	x	x	x	x			
$.getScript(url [, callback])	**					x		
$.getJSON(url [, data][, callback])	**	x					x	
$.get(url [, data][, callback][, dataType])	**	x		x	x	x	x	x
$.post(url [, data][, callback][, dataType])	**		x	x	x	x	x	x
$.ajax(settings)	***	x	x	x	x	x	x	x

Zonder webserver kan u enkel koppelen met statische bestanden. Om gegevens uit een database te verwerken, zal u altijd een server-side script moeten gebruiken en bent u natuurlijk wel verplicht om een webserver te gebruiken.

Omdat niet iedereen vertrouwd is met server-side scripts (*PHP*, *ASP.NET*, ...) gaan we de oefeningen dadelijk opsplitsen over twee verschillende hoofdstukken.

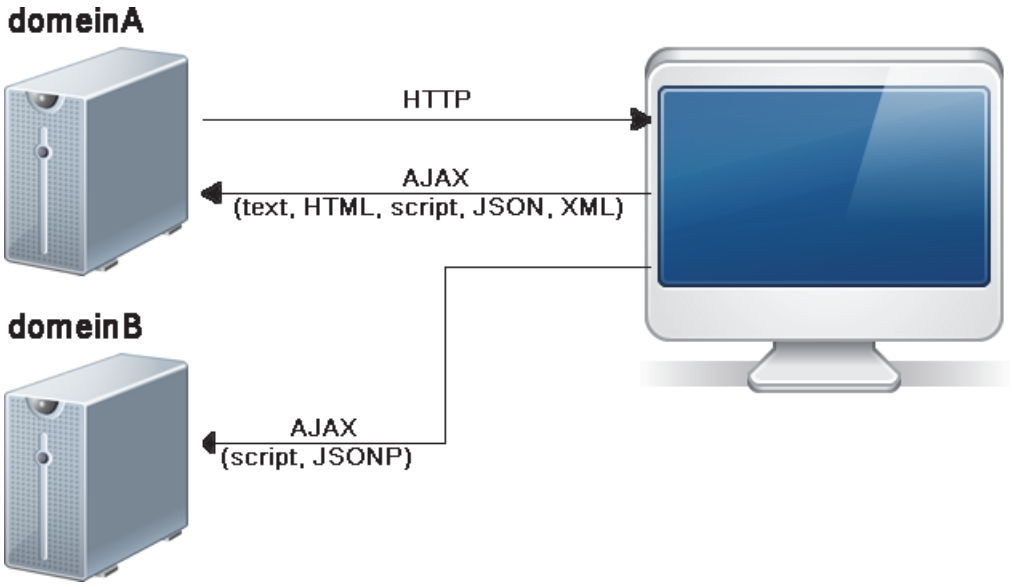
2.6 Same origin policy.

Uit veiligheidsredenen bevatten alle browsers enkele beperkingen. Eén van deze beperkingen is de *same origin policy*. Of, met andere woorden, alle gegevens die u via AJAX ophaalt, moeten afkomstig zijn van hetzelfde domein. Hetzelfde domein betekent meer specifiek: dezelfde server, zelfde protocol, zelfde domeinnaam en dezelfde poort. Het is bijvoorbeeld niet mogelijk dat een webpagina op *domeinA* gegevens ophaalt uit *domeinB*.

Er zijn echter twee uitzonderingen. Het ophalen van externe JavaScripts en gegevens in JSONP formaat (let op de P achteraan) zijn wel toegestaan.

JSONP staat voor "**JSON** with **P**adding". Wanneer de webserver van *domeinB* zodanig staat geconfigureerd dat deze toelaat dat andere gebruikers zijn gegevens in JSON-formaat mogen ophalen, spreekt men van JSONP. De structuur van een JSON-bestand en van een JSONP-bestand is identiek.

Onderstaande afbeelding toont de AJAX mogelijkheden/beperkingen, rekening houdend met de same origin policy.

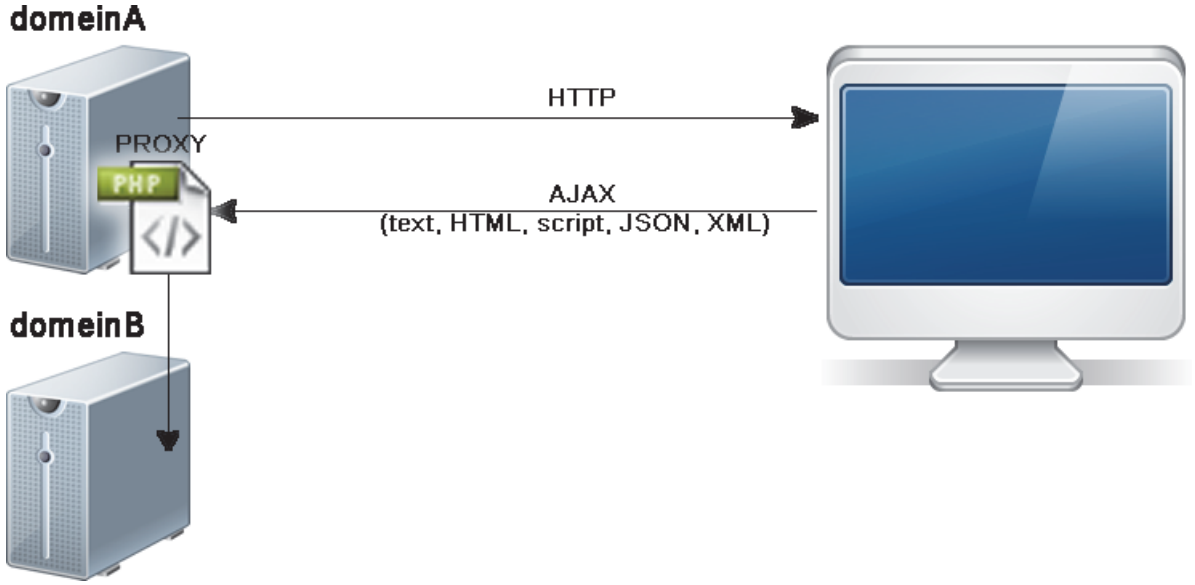


> Same origin policy is een beperking van de browser. Ook indien u zonder webserver werkt (*domeinA* is dan de lokale computer) geldt bovenstaand schema.

2.7 Cross-site scripting.

Indien u over een webserver beschikt, kan u de same origin policy makkelijk omzeilen. Op de webserver plaatst u een proxyscript. Dit proxyscript haalt de inhoud van een externe pagina (html, XML, JSON, ...) op en toont dit in zijn eigen pagina.

In plaats van in een AJAX request te verwijzen naar de externe pagina, verwijst u nu naar de proxypagina. Voor de browser lijkt het alsof de gegevens afkomstig zijn van het eigen domein. Op deze manier kunnen we dus alle externe gegevens perfect via AJAX verwerken. Hierover later meer.



Het proxyscript is een server-side script en is niet gebonden aan de same origin policy.