erasmus

HOGESCHOOL BRUSSEL

Web Development Advanced

erasmus
HOGESCHOOL BRUSSEL

WAT IS AJAX?

Asynchronous JavaScript and XML

Ajax laat je toe…

1

1

Om data op te vragen van de server

1

Om data op te vragen van de server

2

1

Om data op te vragen van de server

2

De data in te laden zonder de pagina te refreshen

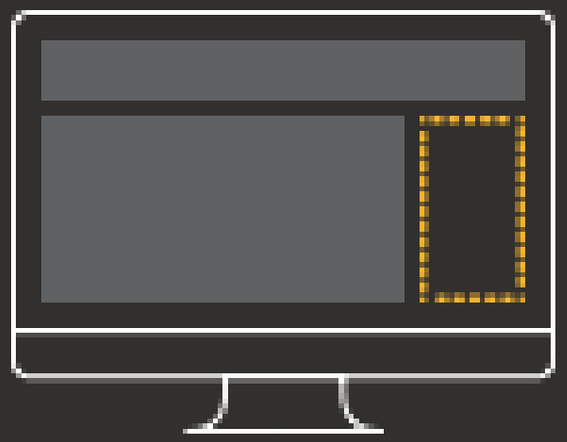
Het maakt gebruik van een asynchroon verwerkings model.

(Gebruikers kunnen andere zaken doen terwijl de data wordt .)

1

DE AANVRAAG

De browser vraagt informatie van de server



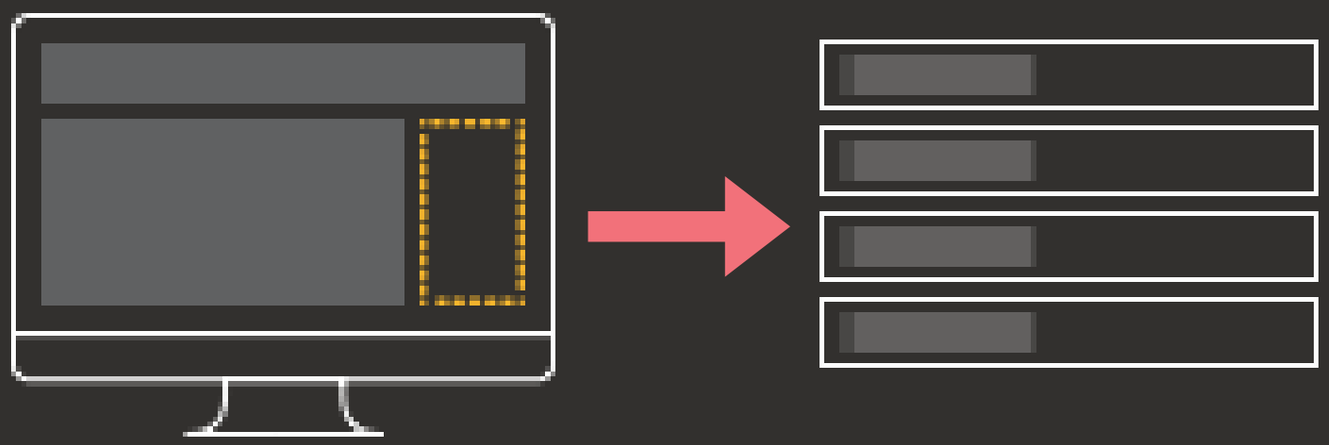
1

DE AANVRAAG

De browser vraagt informatie van de server

OP DE SERVER

De server stuurt data terug (meestal HTML, XML, of JSON)



1

DE AANVRAAG

De browser vraagt informatie van de server

OP DE SERVER

De server stuurt data terug (meestal HTML, XML, of JSON)

2

HET ANTWOORD

De browser verwerkt de inhoud en voegt het toe aan de pagina

REQUEST

(de aanvraag)

Web browsers gebruiken het `XMLHttpRequest` object om de functionaliteiten van Ajax te implementeren.

Hier wordt een instantie van het object opgeslagen in een variabele `xhr`:

```
var xhr = new XMLHttpRequest;
```

De `.open()` method bereidt de aanvraag voor:

```
var xhr = new XMLHttpRequest;
xhr.open('GET', 'test.json', true);
```

Het eerste argument kan ofwel een HTTP `GET` of `POST` zijn:

```
var xhr = new XMLHttpRequest;
xhr.open('GET', 'test.json', true);
```

Het tweede argument specifieert het bestand dat moet worden geladen:

```
var xhr = new XMLHttpRequest;
xhr.open('GET', 'test.json', true);
```

Het derde argument bepaalt of het versoek asynchroon is of niet:

```
var xhr = new XMLHttpRequest;
xhr.open('GET', 'test.json', true);
```

Vervolgens verzenden we de aanvraag:

```
var xhr = new XMLHttpRequest;
xhr.open('GET', 'test.json', true);
xhr.send('search=arduino');
```

RESPONSE

(het antwoord)

Wanneer de server reageerd, roept het `onload` event een anonieme functie op:

```
xhr.onload = function() {
   // process response
};
```

Een eigenschap van het object met de naam `status` wordt vervolgens gebruikt om ervoor te zorgen dat de gegevens goed geladen zijn. :

```
xhr.onload = function() {
   if (xhr.status === 200) {
      // process response
   }
};
```

DATA FORMATS:

HTML

HTML is de eenvoudigste manier om gegevens in een pagina te krijgen:

```
<div class="event">
    <img src=“img/map-ny.png"
         alt="New York, NY” />
    <p><b>New York, NY</b>
    <br>May 30</p>
</div>
```

Het is toegankelijk met de `responseText` property van het object:

```
$el.innerHTML = xhr.responseText;
```

```
var xhr = new XMLHttpRequest();
xhr.onload = function() {
   if(xhr.status === 200) {
      document.getElementByid('content').innerHTML = xhr.responseText;
   }
};
xhr.open('GET', 'data/data.html', true);
xhr.send(null);
```

DATA FORMATS:
XML
(Extensible Markup Language)

XML lijkt op HTML, maar de tags bevatten andere woorden:

```
<event>
    <location>New York, NY</location>
    <date>May 15</date>
    <map>img/map-ny.png</map>
</event>
```

Het is toegankelijk met de `responseXML` property van het object:

```
var events = xhr.responseXML;
```

Je moet een stukje JavaScript schrijven om de XML data om te zetten in HTML, zodat het correct kan weergegeven worden.

```
var xhr = new XMLHttpRequest();

xhr.onload = function() {
  if (xhr.status === 200) {
    var response = xhr.responseXML;
    var events = response.getElementsByTagName('event');
    // zie volgende slide

  }
};

xhr.open('GET', 'data/data.xml', true);
xhr.send(null);
```

```
for (var i = 0; i < events.length ; i++) {
   var container, image, location, city, newline;
   container= document.createElement('div');
   container.className = 'event';

   location = document.createElement('p');
   var value = getNodeValue(events[i],'location');
   var text = document.createTextNode(value);
   location.appendChild(text);

   container.appendChild(location);
   document.getElementByid('content').appendChild(container);
}

function getNodeValue(obj, tag) {
   return obj.getElementsByTagName(tag)[0].firstChild.nodeValue;
}
```

DATA FORMATS:

JSON

(JavaScript Object Notation)

JSON ziet eruit als een object literal syntax maar het is gewoon platte tekst, geen object:

```
{
  “location”: “New York, NY”,
  “date”: “May 30”,
  “map”: “img/map-ny.png”
}
```

Het is toegankelijk met de `responseText` property van het object:

```
var events = xhr.responseText;
```

Je moet een stukje JavaScript schrijven om de JSON data om te zetten in HTML, zodat het correct kan weergegeven worden.

JSON data bestaat uit **keys** en **values**:

```
{
  “location”: “New York, NY”,
  “date”: “May 30”,
  “map”: “img/map-ny.png”
}
```

JSON data bestaat uit **keys** en **values**:

```
{
  “location”: “New York, NY”,
  “date”: “May 30”,
  “map”: “img/map-ny.png”
}
```

De waarde kan een string, nummer, Boolean, array, object of null zijn.

Je kan objecten ook nesten.

```
{
  "events": [
    {
      "location": "Austin, TX",
      "date": "May 15",
      "map": "img/map-tx.png"
    },
    {
      "location": "New York, NY",
      "date": "May 30",
      "map": "img/map-ny.png"
    }
  ]
}
```

```
"events": [
  {
    "location": "Austin, TX",
    "date": "May 15",
    "map": "img/map-tx.png"
  },
  {
    "location": "New York, NY",
    "date": "May 30",
    "map": "img/map-ny.png"
  }
]
```

```
{
    [
      {
        "location": "Austin, TX",
        "date": "May 15",
        "map": "img/map-tx.png"
      },
      {
        "location": "New York, NY",
        "date": "May 30",
        "map": "img/map-ny.png"
      }
    ]
}
```

```
{
  "events": [
    {
      "location": "Austin, TX",
      "date": "May 15",
      "map": "img/map-tx.png"
    },
    {
      "location": "New York, NY",
      "date": "May 30",
      "map": "img/map-ny.png"
    }
  ]
}
```

JavaScript heeft een `JSON` object met twee belangrijke methodes:

1: Omzetten van een JavaScript object naar een string:

```
JSON.stringify();
```

2: Omzetten van een string naar een JavaScript object:

```
JSON.parse();
```

```
var xhr = new XMLHttpRequest();
xhr.onload = function() {
  if(xhr.status === 200) {
    var responseObject = JSON.parse(xhr.responseText);
    var newContent = '';
    for (var i = 0; i < responseObject.events.length; i++) {
       newContent += '<div class="event">';
       newContent += '<p>' + responseObject.events[i].location;
       newContent += '<br/>' + responseObject.events[i].date + '</p>';
       newContent += '</p>';
       newContent += '</div>';
       document.getElementByid('content').innerHTML = newContent ;
    }
  }
};
xhr.open('GET', 'data/data.json', true);
xhr.send(null);
```

JQUERY & AJAX

jQuery biedt methoden aan om om te gaan met Ajax requests / responses:

WERKT OP DE SELECTIE	GLOBALE METHODEN VAN `jQuery` OBJECT
`.load()`	`$.get()`
	`$.post()`
	`$.getJSON()`
	`$.getScript()`
	`$.ajax()`

De `.load()` method geeft de HTML inhoud in, in de jQuery selectie:

```
$('#text').load('ajax.html #text');
```

Het element waar de inhoud zal worden ingeladen:

```
$(‘#text’).load(‘ajax.html #text’);
```

De URL van het bestand komt eerst in het argument:

```
$('#text').load('ajax.html #text');
```

Je kan een fragment van de pagina kiezen die getoond dient te worden (niet de hele pagina):

```
$('#text').load('ajax.html #text');
```

De andere globale Ajax methodes geven hun data terug in `jqxhr` object.

Het `jqxhr` object heeft de volgende properties en methodes:

PROPERTIES

```
responseText
responseXML
status
statusText
```

METHODES

```
.done()
.fail()
.always()
.abort()
```

jQuery heeft 4 shorthand methodes om met specifieke types van ajax requests om te gaan.

`url`	waar de gegevens opgehaald worden
`data`	extra info voor de server
`callback`	functie die aangeroepen wordt bij antwoord server
`type`	type van data dat je verwacht van server

```
$.get(url[, data][, callback][, type])
```

`url`	waar de gegevens opgehaald worden
`data`	extra info voor de server
`callback`	functie die aangeroepen wordt bij antwoord server
`type`	type van data dat je verwacht van server

```
$.get(url[, data][, callback][, type])

$.post(url[, data][, callback][, type])
```

`url`	waar de gegevens opgehaald worden
`data`	extra info voor de server
`callback`	functie die aangeroepen wordt bij antwoord server
`type`	type van data dat je verwacht van server

```
$.get(url[, data][, callback][, type])

$.post(url[, data][, callback][, type])

$.getJSON(url[, data][, callback])
```

`url`	waar de gegevens opgehaald worden
`data`	extra info voor de server
`callback`	functie die aangeroepen wordt bij antwoord server
`type`	type van data dat je verwacht van server

```
$.get(url[, data][, callback][, type])

$.post(url[, data][, callback][, type])

$.getJSON(url[, data][, callback])

$.getScript(url[, callback])
```

Er zijn ook methoden die je helpen omgaan met een Ajax-respons die faalt:

`.done()`	wanneer aanvraag compleet is
`.fail()`	wanneer aanvraag faalt
`.always()`	compleet / faalt

Deze 4 shorthand methodes maken in de achtergrond allemaal gebruik van de `.ajax()` methode.

SETTINGS `.ajax()`

`type`	type request: POST of GET
`url`	Waar de aanvraag naartoe gestuurd wordt
`data`	extra info voor de server
`Succes`	functie die wordt uitgevoerd als Ajax request succesvol was
`Error`	functie die wordt uitgevoerd wanneer er een error optreedt
`beforeSend`	functie die wordt uitgevoerd voordat de Ajax request start
`Complete`	functie die wordt uitgvoerd na success/error event

```
$.ajax ({
  type: "POST",
  url: url,
  beforeSend: function() {
    $content.append('<div id="load">Loading</div>');
  },
  complete: function() {
    $('#loading').remove();
  },
  success: function(data) {
   $content.html($(data).find('#container')).hide().fadeIn(400);
  },
  fail: function() {
    // Toon foutboodschap
  }
});
```

Formulieren verzenden met AJAX

Formulieren worden steeds verzonden met de `.post()` methode.

jQuery biedt de methode `.serialize()` aan om formulier gegevens te verzamelen.

```
$('#register').on('submit', function(e) {
  e.preventOefault();
  var details = $('#register').serialize();
  $.post('register.php', details,
  function(data) {
     $('#register').html(data);
  });
});
```