AJAX

Asynchronous JavaScript and XML

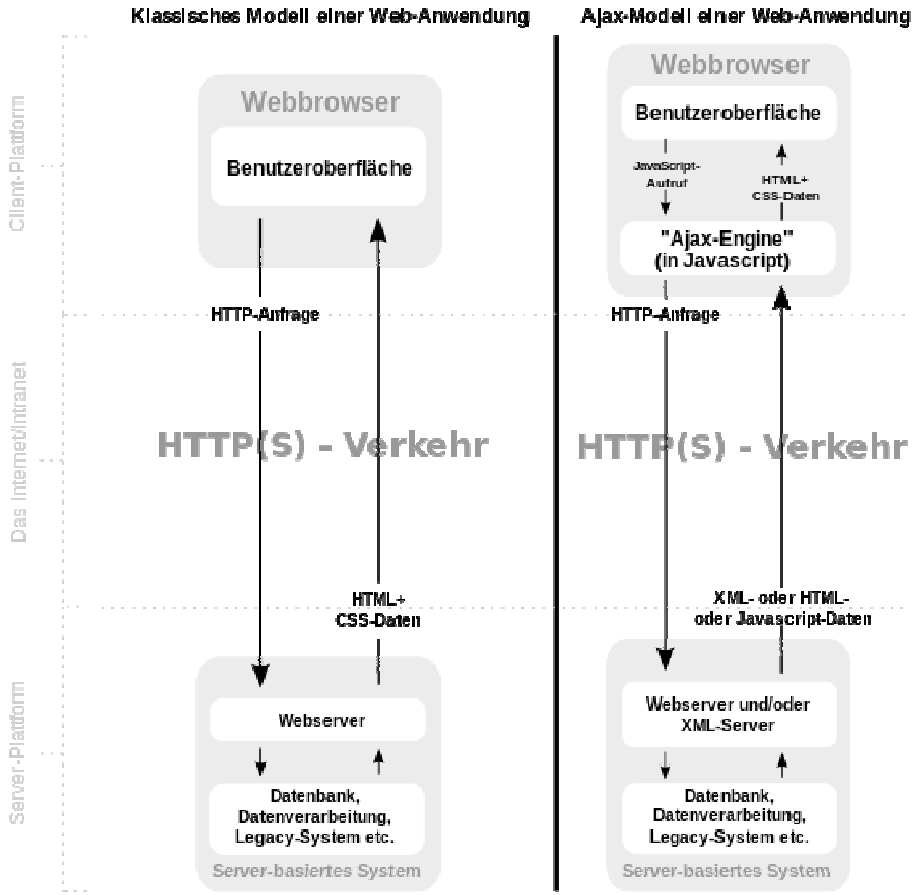
Inhalt

- Definition
- Synchrone vs. Asynchrone Kommunikation
- Datenformate
  - XML-Format
  - JSON-Format
  - Verwendung
- Daten asynchron laden
- Daten asynchron senden

Definition

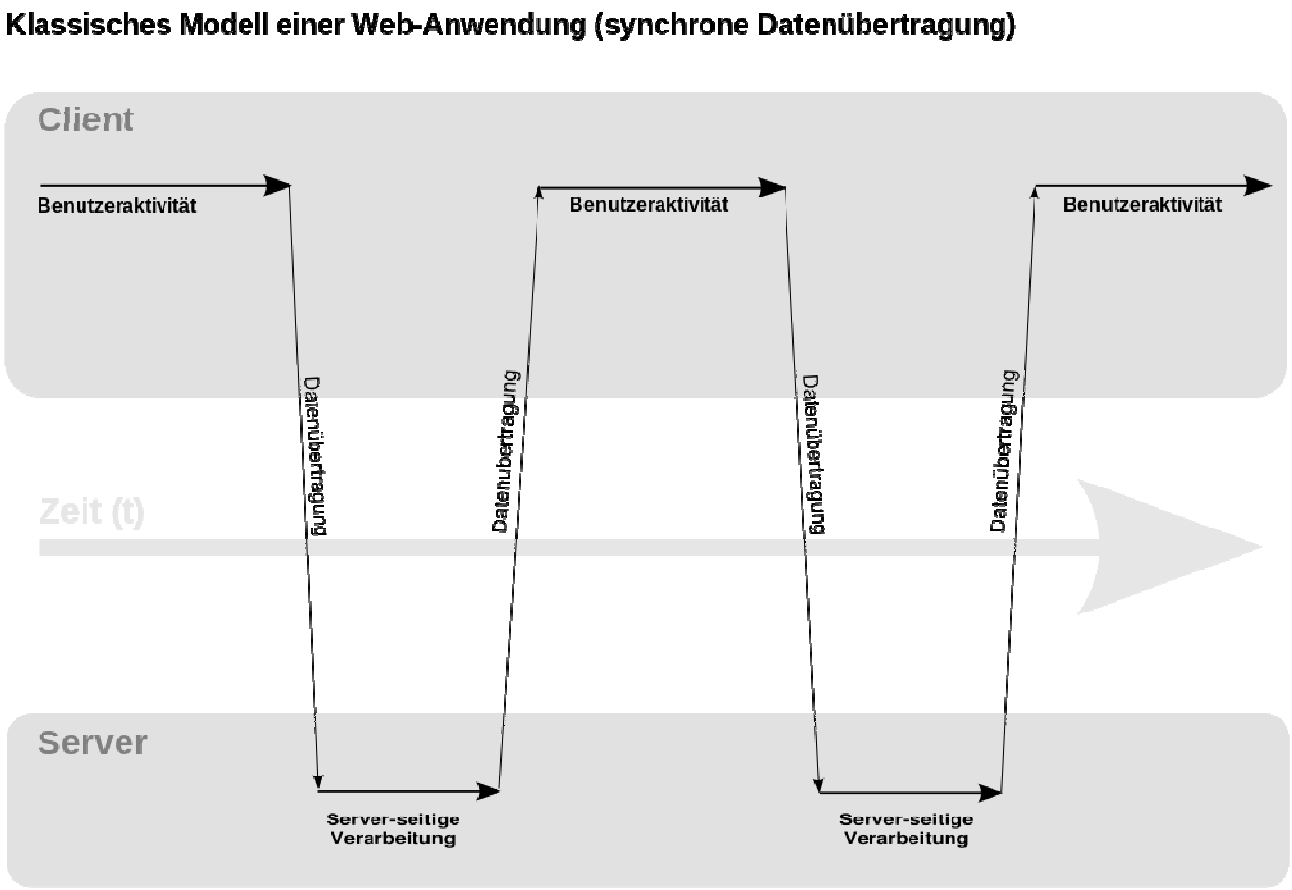
Ist ein Konzept, das es Webanwendungen ermöglicht, neue Daten vom Server zu erhalten und/oder dem Server für die weitere Verarbeitung zu senden, ohne dass die Seite als Ganzes neu geladen wird.

Synchrone vs. Asynchrone Kommunikation



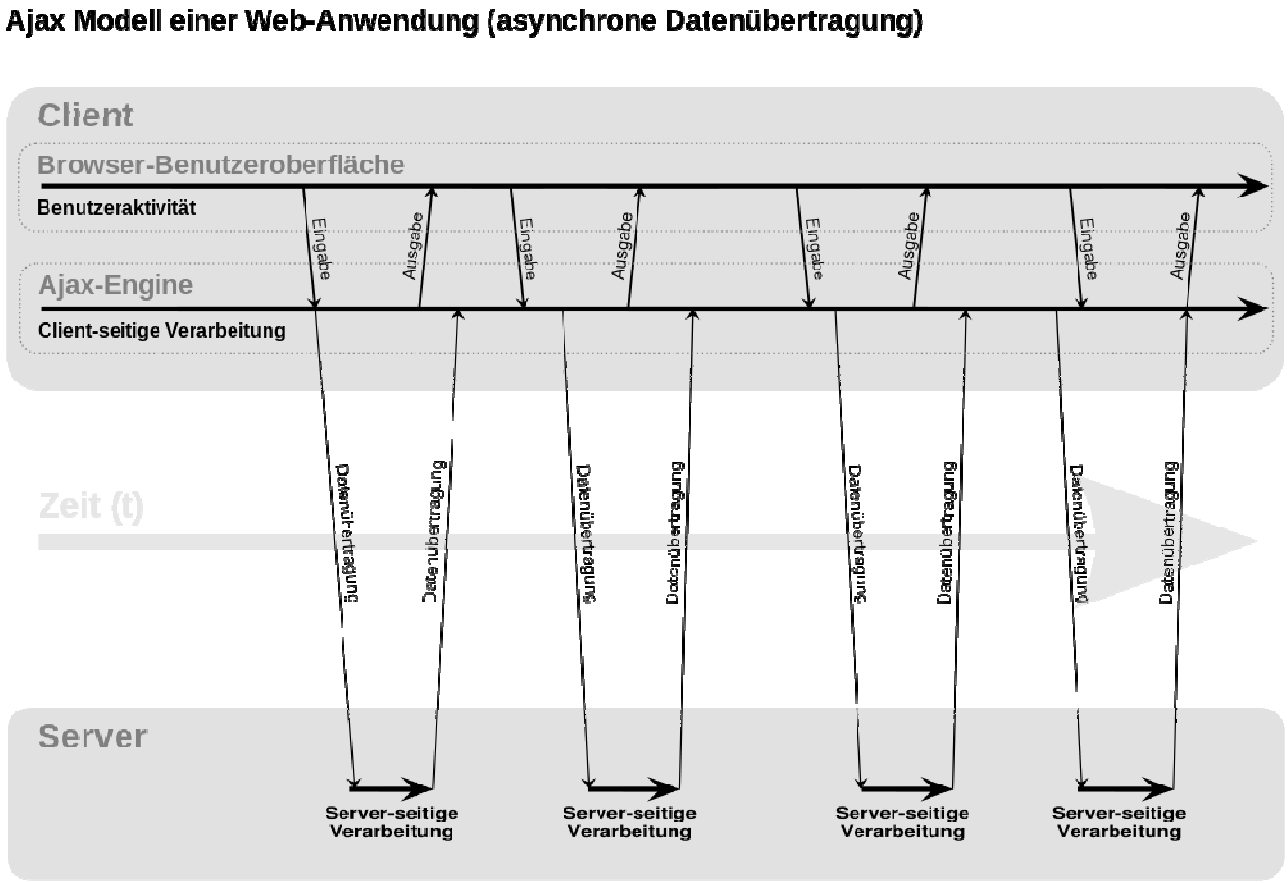
Quelle: https://de.wikipedia.org/wiki/Ajax_(Programmierung)#/media/File:Ajax-vergleich.svg

Synchrone vs. Asynchrone Kommunikation



Quelle: https://de.wikipedia.org/wiki/Ajax_%28Programmierung%29#/media/File:Prozessfluss-traditionell.svg

Synchrone vs. Asynchrone Kommunikation



Quelle: https://de.wikipedia.org/wiki/Ajax_(Programmierung)#/media/File:Ajax-vergleich.svg

Datenformate

Es gibt mehrere Datenformate, die zum Datenaustausch verwendet werden können.

Im AJAX-Umfeld werden folgende Varianten eingesetzt:

- **HTML** – Hypertext Markup Language
- **XML** - Extensible Markup Language
- **JSON** - JavaScript Object Notation

XML-Format

Vorteile

- Flexibel, komplexe Strukturen
- Plattformunabhängig
- Gleiche DOM-Methoden wie bei HTML

Nachteile

- Viel «Overhead»
- z.T. aufwändige Verarbeitung

XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<mitarbeiter>
  <mitarbeiter>
    <vorname>Tony</vorname>
    <nachname>Stark</nachname>
  </mitarbeiter>
  <mitarbeiter>
    <vorname>Peter</vorname>
    <nachname>Parker</nachname>
  </mitarbeiter>
  <mitarbeiter>
    <vorname>Bruce</vorname>
    <nachname>Bana</nachname>
  </mitarbeiter>
</mitarbeiter>
```

XML-Format

Das XML-Format untersteht bestimmten Regeln, die beachtet werden müssen.

Regeln

- Eine XML-Dokument besitzt genau ein Root-Element
- Alle Elemente mit Inhalt besitzen ein Beginn- und End-Tag
- Beginn- und End-Tags korrekt verschachteln
- Attribut-Bezeichnungen müssen pro Element eindeutig sein
- Attribut-Werte müssen in Anführungszeichen stehen
- Gross- und Kleinschreibung wird unterschieden

JSON-Format

Vorteile

- Kompakt, schlank
- Schnell
- Einfacher Aufbau
- Verwendung mit JavaScript

Nachteile

- Strenge Syntax
- Keine Metadaten, Kommentare

JSON

```
{"mitarbeiter":[
  {"vorname":"Tony","nachname":"Stark"},
  {"vorname":"Peter","nachname":"Parker"},
  {"vorname":"Bruce","nachname":"Bana"}
]}
```

JSON vs. XML

Die beiden Datenformate JSON und XML haben einige Gemeinsamkeiten.

1. Sie sind selbstbeschreibend (menschlich lesbar)
2. Sie sind hierarchisch aufgebaut (Werte innerhalb von Werten)
3. Sie können von vielen Programmiersprachen analysiert und genutzt werden

JSON vs. XML

Jedoch bietet JSON einige Vorteile gegenüber XML.

1. Ist kürzer und kompakter
2. Kein grosser «Overhead» (Ballast), der übermittelt wird
3. Ist schneller beim Lesen und Schreiben der Daten
4. Einfachere Verarbeitung der Daten (JavaScript-Parser)

JSON ist optimiert für den Einsatz mit JavaScript. Daher empfiehlt es sich für AJAX-Applikationen JSON statt XML zu verwenden.

XML Parser

Diejenige Software, die XML-Strukturen ausliest, analysiert und nachgeschalteter Software zur Verfügung stellt.

JavaScript

```
var xmlString = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
              "<bookstore><book id=\"978-3-7645-0564-96\">" +
              "<title>Helix</title>" +
              "<author>Marc Elsberg</author>" +
              "<year>2016</year>" +
              "</book></bookstore>";
var parser = new DOMParser();
var xmlDoc = parser.parseFromString(xmlString,"text/xml");
var titleNode = xmlDoc.getElementsByTagName("title")[0].childNodes[0];
console.log(titleNode.nodeValue); // Output: Helix
```

Quelle: https://www.w3schools.com/xml/xml_parser.asp

XML Serializer

Diejenige Software, die XML-Strukturen ausliest, analysiert und nachgeschalteter Software zur Verfügung stellt.

JavaScript

```
var parser = new DOMParser();
var xmlDoc = parser.parseFromString(xmlString,"text/xml");

var xmlSerializer = new XMLSerializer();
var xmlString = xmlSerializer.serializeToString(xmlDoc);
console.log(xmlString);
/* Output: <bookstore><book id="978-3-7645-0564-96">
<title>Helix</title><author>Marc Elsberg</author>
<year>2016</year></book></bookstore> */
```

JSON - Parse

Die Methode `JSON.parse()` parst einen JSON-String in ein JavaScript-Objekt.

JavaScript

```
var jsonString = '{"books":[' +
                 '{"title":"Helix","author":"Marc Elsberg","year":2016},' +
                 '{"title":"Inferno","author":"Dan Brown","year":2013}' +
                 ']}';
var bookstore = JSON.parse(jsonString);
console.log(bookstore.books[1].title); // Output: Inferno
```

Quelle: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse

JSON - Serialisieren

Die `JSON.stringify()` Methode konvertiert einen JavaScript-Wert in einen JSON-String.

JavaScript

```
var book1 = { title:"Helix", author:"Marc Elsberg", year:2016 };
var book2 = { title:"Inferno", author:"Dan Brown", year:2013 };

var bookstore = {"books":[book1, book2] };
var serialisiert = JSON.stringify(bookstore);
console.log(serialisiert);
/* Output: {"books":[
{"title":"Helix","author":"Marc Elsberg","year":2016},
{"title":"Inferno","author":"Dan Brown","year":2013}
]} */
```

Quelle: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify

HTTP-Methoden

Jede Client Request wird durch die Angabe der Methode eingeleitet. Methoden bestimmen die Aktion der Anforderung.

HTTP-Methode	Beschreibung
GET	Anforderung eines Dokuments oder einer anderen Quelle.
POST	Übermittlung von Daten (z.B. Formulareingaben) an den Webserver.
PUT	Modifikation bestehender Quellen bzw. Erzeugung neuer Daten auf dem Server.
PATCH	Ändert ein bestehendes Dokument ohne dieses wie bei PUT vollständig zu ersetzen.
DELETE	löscht die angegebene Ressource auf dem Server.

Quelle: https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol

HTTP-Statuscodes

Jede HTTP-Anfrage wird vom Server mit einem HTTP-Statuscode beantwortet.

Codes	Beschreibung
1xx	Informationen: Die Bearbeitung der Anfrage dauert noch an.
2xx	Erfolgreiche Operation: Die Anfrage war erfolgreich, die Antwort kann verwertet werden.
3xx	Umleitung Um eine erfolgreiche Bearbeitung der Anfrage sicherzustellen, sind weitere Schritte seitens des Clients erforderlich.
4xx	Client-Fehler Die Ursache des Scheiterns der Anfrage liegt (eher) im Verantwortungsbereich des Clients.
5xx	Server-Fehler Die Ursache des Scheiterns der Anfrage liegt (eher) im Verantwortungsbereich des Servers.

Quelle: https://de.wikipedia.org/wiki/HTTP-Statuscode

Daten laden / senden

Das **XMLHttpRequest**-Objekt wird zum Austausch von Daten zwischen Nutzer und Server verwendet. Man kann damit…

- eine Webseite aktualisieren, ohne sie neu laden zu müssen
- Daten vom Server anfordern, nachdem die Seite geladen ist
- Daten zu einem Server im Hintergrund senden

Quelle: https://wiki.selfhtml.org/wiki/JavaScript/XMLHttpRequest

Daten synchron laden

Daten mittels XMLHttpRequest synchron laden.

JavaScript

```
window.addEventListener('load', function load(event) {
  var xhr = new XMLHttpRequest();
  xhr.open('GET', 'data.txt', false); // async = false
  xhr.send();
  document.getElementById('response').innerHTML = xhr.responseText;
});
```

Text

```
Herzlich Willkommen
```

Achtung! JavaScript wird solange warten, bis der Server eine Antwort zurückliefert.

Daten asynchron laden

HTML-Daten mittels XMLHttpRequest asynchron nachladen.

JavaScript

```
window.addEventListener('load', function load(event) {
  var xhr = new XMLHttpRequest();
  xhr.open('GET', 'data.html', true); // async = true
  xhr.setRequestHeader('Accept', 'text/html');
  xhr.onreadystatechange = function () {
    if(xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
      document.getElementById('response').innerHTML = xhr.responseText;
    }};
  xhr.send();
});
```

HTML

```
<p>Das ist ein Absatz, welcher mit AJAX
nachgeladen wurde.</p>
```

Daten asynchron laden

XML-Daten mittels XMLHttpRequest asynchron nachladen.

JavaScript

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'data.xml', true);
xhr.setRequestHeader('Accept', 'text/xml');
xhr.onreadystatechange = function () {
  if(xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
    console.log(xhr.responseXML.getElementsByTagName('absatz'));
  }};
xhr.send();
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
 <absatz>Das ist der erste Absatz</absatz>
 <absatz>Das ist der zweite Absatz</absatz>
 <absatz>Das ist der dritte Absatz</absatz>
</data>
```

Daten asynchron senden

Daten als String an serverseitiges Script (PHP) senden.

JavaScript

```
var data = 'Hallo Server!';
var xhr = new XMLHttpRequest();
xhr.open('POST', 'hello.php', true);
xhr.onreadystatechange = function () {
  if(xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
    console.log(xhr.responseText); // Output: Hallo Client!
  }};
xhr.send(data);
```

PHP

```
<?php
  $data = file_get_contents('php://input'); // Input: Hallo Server!
  echo "Hallo Client!";
?>
```

Daten asynchron senden

Daten im JSON-Format an serverseitiges Script (PHP) senden.

JavaScript

```
var data = { "automarke": ["Audi","VW","Renault"] };
data = JSON.stringify(data); // Konvertierung in einen JSON-String
var xhr = new XMLHttpRequest();
xhr.open('POST', 'send_json.php', true);
xhr.setRequestHeader('Content-Type', 'application/json');
xhr.onreadystatechange = function () {
  if(xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
    console.log(xhr.responseText); // Output: 3
  }};
xhr.send(data);
```

PHP

```
<?php
  $jsonString = file_get_contents('php://input');
  $array = json_decode($jsonString);
  echo count($array->{'automarke'});
?>
```

FormData-Interface

Das FormData-Interface ermöglicht das einfache Erstellen von Schlüssel/Werte-Paaren, welche Formular-Daten repräsentieren. Es kann leicht durch aufrufen der `XMLHttpRequest.send()` Methode abgeschickt werden. Das benutze Format ist identisch zu einem HTML-Formular dessen encoding-Typ auf "multipart/form-data" gesetzt wurde.

Quelle: https://developer.mozilla.org/de/docs/Web/API/FormData

Daten asynchron senden

Daten im Formular-Format an serverseitiges Script (PHP) senden.

JavaScript

```
var data = new FormData();
data.append('Nachname', 'Bond');
data.append('Vorname', 'James');
var xhr = new XMLHttpRequest();
xhr.open('POST', 'form_data.php', true);
xhr.setRequestHeader('Content-Type', 'multipart/form-data');
xhr.onreadystatechange = function () {
  if(xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
    console.log(xhr.responseText); // Output: Hello James Bond
  }};
xhr.send(data);
```

PHP

```
<?php echo 'Hello '.$_POST['Vorname'].' '.$_POST['Nachname']; ?>
```

Daten asynchron senden

Daten über ein HTML-Formular an serverseitiges Script (PHP) senden.

JavaScript

```
var form = document.getElementById('formular');
form.addEventListener('submit', function(e) {
  e.preventDefault(); // Standardaktion wird abgebrochen
  var data = new FormData(form);
  var xhr = new XMLHttpRequest();
  xhr.open('POST', 'form_data.php', true);
  xhr.send(data);
};
```

HTML

```
<form id="formular" method="post">
  <input type="text" name="infos">
  <input type="submit" name="submit">
</form>
```

Online-Hilfe

w3schools.com

https://www.w3schools.com/js/js_ajax_intro.asp

<!selfhtml />

https://wiki.selfhtml.org/wiki/JavaScript/Ajax