

MICHAEL REICHART

DIE JAVASCRIPT-  
BIBLIOTHEK JQUERY

DER AUTOR VON JQUERY  
JOHN RESIG

# JQUERY

- John Resig entwickelt die jQuery Klassenbibliothek und stellt sie im Januar 2006 auf dem BarCamp in New York vor.
- Seitdem wird jQuery laufend weiterentwickelt.
- Bereits im September 2008 haben Microsoft und Nokia angekündigt, jQuery in ihren Produkten zu verwenden.
- Heute ist jQuery die meistverwendete Javascript Bilbiothek der Welt.
- Quelle: Wikipedia, Bild von Microsoft



# JQUERY

- jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

# JQUERY 1.12.X

- Lightweight Footprint  
Only 32kB minified and gzipped. Can also be included as an AMD module (Die Entwicklerversion hat rund 250 kB.)
- CSS3 Compliant  
Supports CSS3 selectors to find elements as well as in style property manipulation
- Cross-Browser  
IE, Firefox, Safari, Opera, Chrome, and more

# JQUERY 3.X FOR HTML5 - BROWSER ONLY

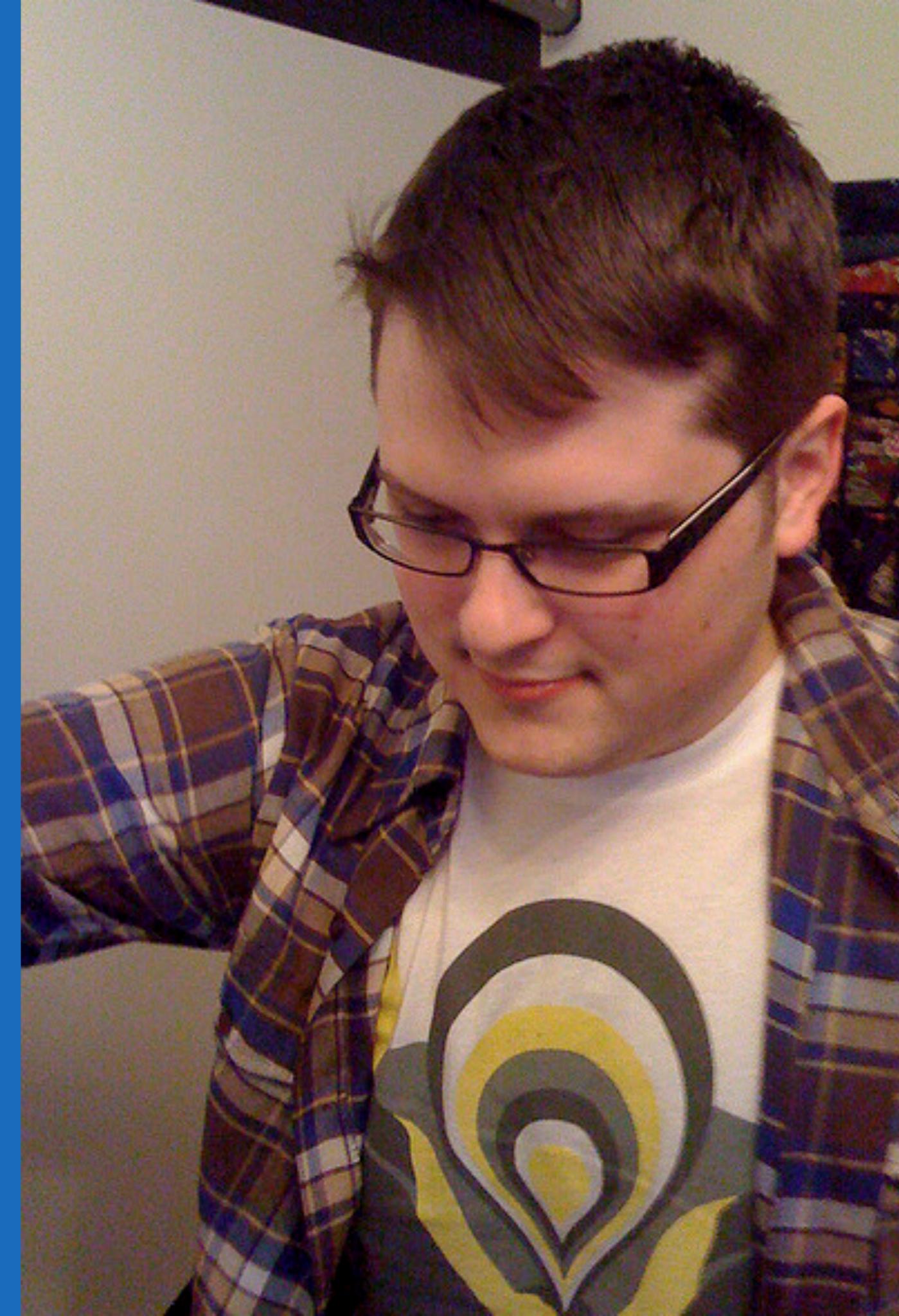
- Mit HTML5 fallen viele Vorteile von Jquery zunächst einmal weg: Animationen, erweitertes Eventhandling, Traversen und die Selectorengine Sizzle werden von nativen Javascript oder CSS Features ausgeführt.
- Die jQuery Foundation entwickelt daher mit den Versionsnummern 2.x einen Softwarezweig für HTML5 Browser. Diese wird auf die langsamere Nachbildung der oben genannten Funktionen verzichten und auf die nativen Fähigkeiten der Browser bauen.

# JQUERY MIGRATION

- jQuery 1.12.x wird weiterhin und solange für alle 'alten' Browser weiterentwickelt, solange diese relevant sind.
- jQuery 3.0.0 für die 'neuen' Browser.
- Zudem bietet die jQuery Foundation ein Migrations-Skript an, das Hinweise zur Migration alter auf neue jQuery Versionen gibt.

# ANDERE PROJEKTE VON JOHN RESIG

- **processing.js**  
A port of the Processing visualisation language in Javascript
- **sizzle.js**  
Javascript selector library
- **QUnit**  
Javascript test suite and testrunner
- **FireUnit**  
Javascript unittesting for Firefox
- **Simple Inheritance**  
Simple class style Javascript inheritance
- ...
- <http://ejohn.org/>



# INSTALLATION



Plugins Contribute Events Support jQuery Foundation



Your donations help fund the continued development and growth of jQuery.

[SUPPORT THE PROJECT](#)

Download API Documentation Blog Plugins Browser Support

Search



### Lightweight Footprint

Only 32kB minified and gzipped. Can also be included as an AMD module



### CSS3 Compliant

Supports CSS3 selectors to find elements as well as in style property manipulation



### Cross-Browser

Chrome, Edge, Firefox, IE, Safari, Android, iOS, and more

## Download jQuery

V3.4.1

[View Source on GitHub →](#)

[How jQuery Works →](#)

## What is jQuery?

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

## Corporate Members

## Resources

- [jQuery Core API Documentation](#)
- [jQuery Learning Center](#)
- [jQuery Blog](#)
- [Contribute to jQuery](#)
- [About the jQuery Foundation](#)
- [Browse or Submit jQuery Bugs](#)

## Downloading jQuery

Compressed and uncompressed copies of jQuery files are available. The uncompressed file is best used during development or debugging; the compressed file saves bandwidth and improves performance in production. You can also download a [sourcemap file](#) for use when debugging with a compressed file. The map file is *not* required for users to run jQuery, it just improves the developer's debugger experience. As of jQuery 1.11.0/2.1.0 the `//# sourceMappingURL` comment is [not included](#) in the compressed file.

To locally download these files, right-click the link and select "Save as..." from the menu.

### jQuery

For help when upgrading jQuery, please see the [upgrade guide](#) most relevant to your version. We also recommend using the [jQuery Migrate plugin](#).

[Download the compressed, production jQuery 3.1.1](#)

[Download the uncompressed, development jQuery 3.1.1](#)



[Download the map file for jQuery 3.1.1](#)

You can also use the slim version:

[Download the compressed, production jQuery 3.1.1 slim build](#)

[Download the uncompressed, development jQuery 3.1.1 slim build](#)

[Download the map file for the jQuery 3.1.1 slim build](#)

[jQuery 3.1.1 release notes](#)

## Downloading jQuery using npm

jQuery is registered as [a package](#) on [npm](#). You can install the latest version of jQuery with the command:

```
1 | npm install jquery
```

# JQUERY 3 SLIM

- Keine Effekte, kein Ajax, kein veralteter Code:
- Entfernt sind:  
jQuery.fn.extend, jQuery.fn.load, jQuery.each,  
jQuery.expr.filters.animated, jQuery.ajaxSettings.xhr,  
jQuery.ajaxPrefilter, jQuery.ajaxSetup,  
jQuery.ajaxPrefilter, jQuery.ajaxTransport,  
jQuery.ajaxSetup, jQuery.parseXML, jQuery.easing,  
jQuery.Animation, jQuery.speed

# JQUERY FÜR LEGACY BROWSER

- Ab Version 2 hat jQuery die Unterstützung für alte Browser ausgelagert.
- Wer für IE 6, 7 oder 8 entwickelt, muss jQuery 1.12. x verwenden.

# JQUERY MIGRATE PLUGIN

- Das migrate-Plugin vereinfacht die Anpassung von Skripten in alten Versionen zu neuen. Es weist auf deprecated-Funktionen und Verhaltensweisen hin.
- jQuery Migrate 1.4.1 für 1.x nach 3.x
- jQuery Migrate 3.0.0 für 2.x nach 3.x

# USING JQUERY WITH A CDN

```
<script  
  src=„http://code.jquery.com/  
    jquery-3.4.1.min.js“  
></script>  
  
<script  
  src=„http://code.jquery.com/  
    jquery-migrate-1.4.1.min.js“  
></script>  
  
<script  
  src=„http://code.jquery.com/  
    jquery-migrate-3.0.1.min.js“  
></script>
```

# EINBINDUNG IN HTML

# JQUERY UND EIGENE SKRIPTE EINBINDEN

```
<html>
  <head> ... </head>
  <body>
    ...
    ...
    ...
    <script
      src=„http://code.jquery.com/
          jquery-3.1.1.min.js“
    ></script>
    <script
      src=„http://code.jquery.com/
          jquery-migrate-3.0.0.js“
    ></script>
    <script
      src=„sources/scripts/
          my-own-code.js“
    ></script>
  </body>
</html>
```

# BEISPIELE FÜR JQUERY

# DOM MANIPULATION

```
$( "button.continue" ).html( "Next Step...." );
```

# EVENT HANDLING

```
var hiddenBox = $( "#banner-message" );  
  
$( "#button-container button" )  
  .on(  
    "click",  
    function(event) {  
      hiddenBox.show();  
    }  
);
```

# AJAX FUNCTIONS

```
$ajax({  
  url : "/api/getWeather",  
  data : {  
    zipcode: 97201  
  },  
  success : function( data ) {  
    $( "#weather-temp" )  
      .html( „<strong>“+data  
            +"  
            "</strong> degrees" );  
  }  
});
```

# \$() ODER JQUERY()?

# `$(), JQUERY()`

- `$` und `jQuery` sind Synonyme.
- Die Funktion `$()` ist die `jQuery` Factoryfunktion.
- In der Factoryfunktion ist das gesamte Dokument als Objekt abgebildet.
- Auf ihr werden alle `jQuery` Befehle ausgeführt.

# \$(), JQUERY()

```
$('p').addClass('myClass');  
jQuery('p').addClass(...)  
$.ajax();  
jQuery.ajax();
```

# JQUERY SAMMELT OBJEKTE EIN - COLLECTION ERZEUGEN

- Das jQuery - Object umgibt jQuery Code als Wrapper.
- Der Selector extrahiert aus dem DOM eine Sammlung von Elementen, Collection genannt.
- Eine Befehlskette wird auf jedes Element der Collection angewandt.

# COLLECTION - EIN ARRAY AUS OBJEKTEN

```
$('nav-main li').addClass('active');

$('nav-main li') => [
    [li],
    [li],
    [li]
]
```

# `$(), JQUERY()`

- \$ wird auch von anderen Frameworks als Objekt verwendet: Prototype, Zepto, Ext.js ...
  - Dann jQuery als Wrapper verwenden.
  - Oder mit `noConflict()` eine eigene Variable erfinden
- ...

# ANDERER WRAPPER ODER MEHRERE JQUERY VERSIONEN?

```
// noConflict!
<script src='jquery-1.12.1.js'></script>
<script>
  var jQuery1121 = jQuery.noConflict();
  jQuery1121('selector') . ....
</script>

<script src='jquery-3.0.0.js'></script>
<script>
  var jQuery300 = jQuery.noConflict();
  jQuery300('selector') . ....
</script>
```

`$DOCUMENT.READY()`

# `$(DOCUMENT).READY()`

- `.ready()` ist eine Methode aus dem jQuery Eventmodul. Eine Funktion, die an `.ready()` übergeben wird, führt jQuery aus, sobald das Dokument geladen wird.
- Funktionen werden erst ausgeführt, wenn der HTML DOM geladen ist, dies verhindert Javascript Fehler aufgrund unvollständigen DOMs.

# AUSFÜHREN EINER ANONYMEN FUNKTION ALS CALLBACK ...

```
$(document).ready( function() {  
    $("#box p").addClass("green");  
});
```

```
// Alternativ wird auch verwendet  
$(function () { ... })
```

# WINDOW.ONLOAD

```
window.onload = function () { ... }
```

=> wird erst am Ende aktiv, wenn A L L E S geliefert und verarbeitet wurde. Es gibt nur einen window.onload Event.

# JQUERY'S DOM-READY ALTERNATE

```
$function () { ... };
```

=> Dies scheint eine selbstausführende Funktion als Callback eines `$document.ready()` Events zu sein.

# IMMEDIATE FUNCTION (SELBSTAUSFÜHRENDE FUNKTION)

```
(function () { ... })();
```

=> wird vor allen anderen Funktionen geladen,  
ggf. zu früh für DOM-Manipulationen

# SELBSTAUSFÜHRENDE FUNKTION UND JQUERY'S DOM-READY

```
// Define "MyApp" as a revealing module
MyApp = (function(Backbone, $){

  var View = Backbone.View.extend({
    // do stuff here
  });

  return {
    init: function(){
      var view = new View();
      $("#some-div").html(view.render().el);
    }
  };
})(Backbone, jQuery);
```

MODULE-BLOCK PATTERN

```
// Run "MyApp" in DOMReady
$(function(){
  MyApp.init();
});
```

Unobtrusivismn.

– JOHN RESIG

# DAS MANTRA DES REINEN HTML ...

- ist die strikte Trennung von Inhalt/Struktur, Design und Verhalten/Logik.
- HTML - Inhalt und Struktur des Inhalts
- Stylesheets - Design und Designverhalten
- Javascript - Verhalten und clientseitige Logik

# OBTRUSIVE PROGRAMMING

```
<div id="box">
  <p onclick="farbwechsel(this)">
    Erster Absatz
  </p>
  <p onclick="farbwechsel(this)">
    Zweiter Absatz
  </p>
  <p onclick="farbwechsel(this)">
    Dritter Absatz
  </p>
</div>
```

# UNOBTUSIVE PROGRAMMING

```
$(document).ready(function() {  
    $("#box p").click(function() {  
        $(this).addClass("green");  
    });  
});  
  
<div id="box">  
    <p>Erster Absatz</p>  
    <p>Zweiter Absatz</p>  
    <p>Dritter Absatz</p>  
</div>
```

# DIE RICHTIGE DOM - ARCHITEKTUR

- Objekte anlegen!
- Objekte benennen!
- Objekte formatieren!
- Für Javascript/jQuery und Stylesheets lassen sich in der Regel dieselben Selektoren verwenden. Daher lohnt sich die Entwicklung einer sauberen, zu Ende gedachten DOM Struktur einer Webseite oder Applikation.

# OBJEKTE ANLEGEN

```
<html>  
<head>...</head>  
<body>  
  <div>  
    ...  
  </div>  
</body>  
</html>
```

NICHT SEMANTISCHER CONTAINER ZUR POSITIONIERUNG

# OBJEKTE ANLEGEN

```
<html>  
<head>...</head>  
<body>  
  
    <div>  
        <nav>  
            ...  
        </nav>  
    </div>  
  
</body>  
</html>
```

SEMANTISCHER CONTAINER FÜR DIE  
DOKUMENTENSTRUKTUR

# OBJEKTE ANLEGEN

```
<html>
<head>...</head>
<body>

    <div>
        <nav>
            <ul>
                <li><a href=„url.html“>Linktext</a></li>
            </ul>
        </nav>
    </div>

</body>
</html>
```

SEMANTISCHE STRUKTUR DER INHALTE

# OBJEKTE ANLEGEN

```
<html>
<head>...</head>
<body>

    <div>
        <nav>
            <ul>
                <li><a href=„url.html“>Linktext</a></li>
            </ul>
        </nav>
    </div>

</body>
</html>
```

# OBJEKTE BENENNEN

```
<html>
<head>...</head>
<body>

    <div>
        <nav id=„nav-main“>
            <ul>
                <li><a href=„url.html“>Linktext</a></li>
            </ul>
        </nav>
    </div>

</body>
</html>
```

# OBJEKTE FORMATIEREN

```
<html>
<head>...</head>
<body>
    <div class=„page-navigation“>
        <nav id=„nav main“>
            <ul class=„nav nav-pills nav-stacked“>
                <li><a href=„url.html“>Linktext</a></li>
            </ul>
        </nav>
    </div>
</body>
</html>
```

# SELEKTIEREN

```
<div class=„page-navigation“>
  <nav id=„nav-main“>
    <ul class=„nav nav-pills nav-stacked“>
      <li><a href=„url.html“>Linktext</a></li>
    </ul>
  </nav>
</div>
```

.page-navigation { ... } -> Templateklasse

.nav, .nav li { ... } -> allg. Basisklassen für Nav-Objekt

.nav-pills { ... } -> allg. Designklasse für Nav-Objekt

.nav-stacked { ... } -> allg. Layoutklasse für Nav-Objekt

#nav-main { ... } -> Einheitlicher Selektor für Elemente  
der Hauptnavigation

JQUERY SCHAFFT VEREINFACHUNG!  
VERGLEICH MIT NORMALEM  
JAVASCRIPT

# DER ZWEITE TEXTABSATZ SOLL GRÜN WERDEN

```
// HTML
<div id="box">
  <p>Erster Absatz</p>
  <p>Zweiter Absatz</p>
  <p>Dritter Absatz</p>
</div>

// CSS
.green {color:#009933; }

// jQuery
$("#box p:contains('Zweiter Absatz')").addClass("green");

// Javascript
var elements = document.getElementById("box").getElementsByTagName("p");

for (var i = 0; i < elements.length; i++) {
  if (elements[i].firstChild.data == "Zweiter Absatz") {
    elements[i].className = "green";
  }
}
```

# EVENTS IN MSIE UND W3 BROWSERN ERZEUGEN VS. JQUERY UNIFIED METHOD

```
// In den Standardbrowsern nach W3
myP[i].onClick = function(event) {
    alert('Klick an ' + event.clientX + ',' + event.clientY);
}

// und im Internet Explorer < 9
myP[i].onClick = function() {
    alert('Klick an ' + event.clientX + ',' +
        event.clientY);
}

// jQuery's einheitliche Syntax:
$(document).ready(function(){
    $('p').click( function(event) { ... });
});
```

# DIE SELEKTORENGINE "SIZZLE" ALS BASIS ZUR DOM-MANIPULATION

# BASIS SELEKTOREN

- p => Elementeselektor
  - #box div#box => ID–Selektoren (frei, gebunden)
  - .nav nav.nav => Klassenselektoren (frei, geb.)
- Attributselektoren:**
- [title] a[href] input[type=email]

# DOM SELEKTOREN

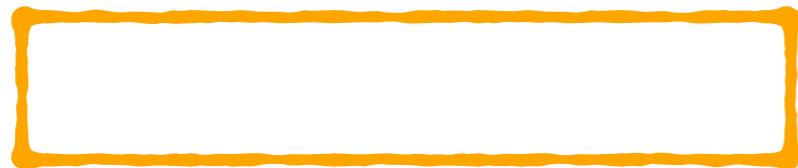
- ul li => Descendant (Nachfahren-) Selektor
- div > nav => Child (Kind) -Selektoren
- li ~ li => General Following Sibling (Geschwister)
- label + [type=text] => Immediate Following Sibling

# ELEMENTVERWANDTSCHAFTEN IM DOM BAUM

ancestor of e

parent of e

previous sibling of e



child of e

descendant of e

sibling of e (immediate following)

sibling of e (following)

# SELEKTORENAUSDRÜCKE

- `$(document)`      => DOM Objekt als Selektor
- `$( 'body' ), $( '#box' ), $( '.nav' )`      => CSS-Selektoren
- `$( ':empty' ), $( ':hidden' )`      => jQuery-Selektoren

# DIE COLLECTION ALS OBJEKT SPEICHERN

```
// Eine Collection als Objekt speichern
var obj = $('#container');

// jQuery Methoden auf das Objekt anwenden
obj.addClass('classname')

// Meistens nicht nötig, aber möglich
// Objekt als Selektorausdruck
$(obj).addClass('classname')
```

# CSS SELEKTOR MIT KONTEXT

```
$("#id", "form#contact")  
  
<form id="contact">  
  <p id="id"> ... </p>  
</form>  
  
// === $('form#contact #id')
```

# GRUPPENSELEKTOR

```
$('p, ul, li')

<h1> ...
<p> ...
<hr>
<ul>
  <li> ...
</ul>
<footer> ... </footer>
```

# MEHRFACHKLASSENSELEKTOR

```
$( '.myClassA.myClassB' );  
  
<p class="myClassA myClassB"> ... </p>
```

Aber nicht:

```
<p class="myClassA"> ... </p>  
<p class="myClassB"> ... </p>
```

# SELEKTOREN-FILTER

`$("div p:first");`      => jQuery – Schreibweise

`$("div p:first-child");`    => CSS 3 – Schreibweise

```
<div>
  <p> ... </p>
  <p> ... </p>
  <p> ... </p>
</div>
```

# SELEKTOREN-FILTER

`$("table tr:even");` => jQuery – Schreibweise

`$("table tr:nth-child(even)");` => CSS 3 – Schreibweise

```
<table>
  <tr> ...
  <tr> ...
  <tr> ...
  <tr> ...
  <tr> ...
</table>
```

# SELEKTOREN-FILTER

:first	Das erste Element des übergebenen Typs
:last	Das letzte Element des übergebenen Typs
:even	Elemente mit geraden Index
:odd	Elemente mit ungeradem Index
:not(sel)	Elemente, auf die der Selektor nicht zutrifft
:eq(n)	Element der Collection mit dem Index n
:lt(n)	Elemente mit einem Index kleiner als n
:gt(n)	Elemente mit einem Index größer als n
:animated	Elemente, die gerade animiert werden
:header	Elemente, die Überschriften h1, h2 ... entsprechen

# WEITERE FILTER ...

## Inhaltsfilter

:contains(text)	=> enthält einen Text
:empty	=> ist leer
:has(sel)	=> besitzt mindestens ein Element mit (sel)
:parent	=> gibt das Elternelement zurück

## Sichtbarkeitsfilter

:visible	=> Elemente mit display:block; inline-block ...
:hidden	=> Elemente mit display:none;

# SETZEN UND ABFRAGEN VON DOM ELEMENTEN UND ATTRIBUTEN

# INSERTION INSIDE FUNCTIONS

## **.append()**

Insert content, specified by the parameter, to the end of each element in the set of matched elements.

## **.appendTo()**

Insert every element in the set of matched elements to the end of the target.

## **.prepend()**

Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

## **.prependTo()**

Insert every element in the set of matched elements to the beginning of the target.

## **.text()**

Get the combined text contents of each element in the set of matched elements, including their descendants, or set the text contents of the matched elements.

# INSERTION OUTSIDE FUNCTIONS

## **.after()**

Insert content, specified by the parameter, after each element in the set of matched elements.

## **.before()**

Insert content, specified by the parameter, before each element in the set of matched elements.

## **.insertAfter()**

Insert every element in the set of matched elements after the target.

## **.insertBefore()**

Insert every element in the set of matched elements before the target.

# .APPEND() FÜGT ELEMENTE EIN

```
$('container').append('p')
```

=> Schreibt ein neues p-Element in den #container.  
Das Ergebnis ist sofort sichtbar.

# .APPENDTO() KONSTRUIERT ELEMENTE!

```
$( '<p>' ).appendTo( '#container' )
```

```
$( '<p>', frames[n].document ).appendTo( 'body' )
```

=> \$( '<p>' ) schreibt ein p-Element in den Speicher.  
Hier lassen sich weitere Manipulationen durchführen.  
Am Ende wird es im DOM verankert.

```
$( '<p>' ).html('Text darin').css('color',  
'red').appendTo('body');
```

# .ADDCLASS()

```
$( "p" ).addClass( "green" );
```

=> Fügt eine Klasse zu den Elementen einer Collection hinzu.  
Dabei wird das Klassenattribut gesetzt, falls es noch keines  
gibt, ansonsten wird die Klasse zu bestehenden hinzugefügt.

```
<p class="green">  
<p class="old green">
```

# .REMOVECLASS()

```
$("p").removeClass("green");
```

=> Löscht eine Klasse aus den Elementen einer Collection.  
Dabei wird das Klassenattribut nicht entfernt und bleibt  
gegebenenfalls leer zurück.

```
<p class="green"> // Vorher  
<p class="">      // Nachher
```

# .HASCLASS()

```
var obj = $("p");
if (obj.hasClass("green")){
  obj.addClass("greenBorder");
}
```

Frägt die Collection, ob eine Klasse vorhanden ist oder nicht und liefert ein boolsches Ergebnis.

# .toggleClass()

```
$("p").toggleClass("green");
```

=> Schaltet zwischen einer Klasse und dem Ausgangszustand hin und her. Dabei wird das Klassenattribut gesetzt, falls es noch keines gibt, ansonsten wird die Klasse zu bestehenden hinzugefügt.

<p class="old">	// Vorher
<p class="old green">	// Nachher
<p class="old">	// Zweiter Klick
	// usw.

## .ATTR()

```
$( 'elem' ).attr('class');           // Abfrage  
$( 'elem' ).attr('class', 'value'); // Setzen
```

=> Um ein Attribut abzufragen oder einzufügen, wird attr() verwendet. Beim Setzen eines Attributes werden vorhandene ggf. überschrieben.

# .REMOVEATTR()

```
$('p').removeAttr('class');
```

löscht ein Attribut, samt Key und Value.

```
<p class="old">      // Vorher  
<p>                  // Nachher
```

# .css()

## .css()

Get the value of a style property for the first element in the set of matched elements or set one or more CSS properties for every matched element.

```
.css({  
  border : '1px solid red',  
  color  : 'black'  
})
```

Fügen Sie dem Formular eine Checkbox mit der Beschriftung "keep password" hinzu.

```
<div id="keep">  
  <input type="checkbox" id="cb">  
  <label for="cb">keep Password</label>  
</div>
```

AUFGABE

# INHALTE VON ELEMENTEN BEARBEITEN

# .HTML()

```
$('p').html('Lorem <b>ipsum</b> dolor sit amet.')
```

=> Fügt einen HTML Knoten in ein DOM Element ein.

  Lorem ipsum dolor sit amet.

# .TEXT()

```
$('p').text('Lorem <b>ipsum</b> dolor sit amet.');
```

=> Fügt einen Textknoten in ein DOM Element ein.  
Dabei werden Sonderzeichen 'HTML-escaped'

  Lorem &lt;b&gt;ipsum&lt;/b&gt; dolor sit amet.

# .VAL()

```
$(“input#email”).val(“michael@zenbox.de”)
```

=> Liest oder setzt einen Wert aus einem Formularfeld.  
Eine Funktion die sämtliche Werte aus einem Formular ausliest,  
gibt es auch.

```
var arr = $('form').serializeArray();
```

# ERSETZEN

## **.replaceAll()**

Replace each target element with the set of matched elements.

## **.replaceWith()**

Replace each element in the set of matched elements with the provided new content and return the set of elements that was removed.

# ELEMENTE EINFÜGEN

## **.prepend()**

Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

## **.prependTo()**

Insert every element in the set of matched elements to the beginning of the target.

## **.after()**

Insert content, specified by the parameter, after each element in the set of matched elements.

## **.before()**

Insert content, specified by the parameter, before each element in the set of matched elements.

## **.insertAfter()**

Insert every element in the set of matched elements after the target.

## **.insertBefore()**

Insert every element in the set of matched elements before the target.

## **.clone()**

Create a deep copy of the set of matched elements.

# PARENT ELEMENTE

## **.unwrap()**

Remove the parents of the set of matched elements from the DOM, leaving the matched elements in their place.

## **.wrap()**

Wrap an HTML structure around each element in the set of matched elements.

## **.wrapAll()**

Wrap an HTML structure around all elements in the set of matched elements.

## **.wrapInner()**

Wrap an HTML structure around the content of each element in the set of matched elements.

# ELEMENTE/INHALTE LÖSCHEN

## **.detach()**

Remove the set of matched elements from the DOM.

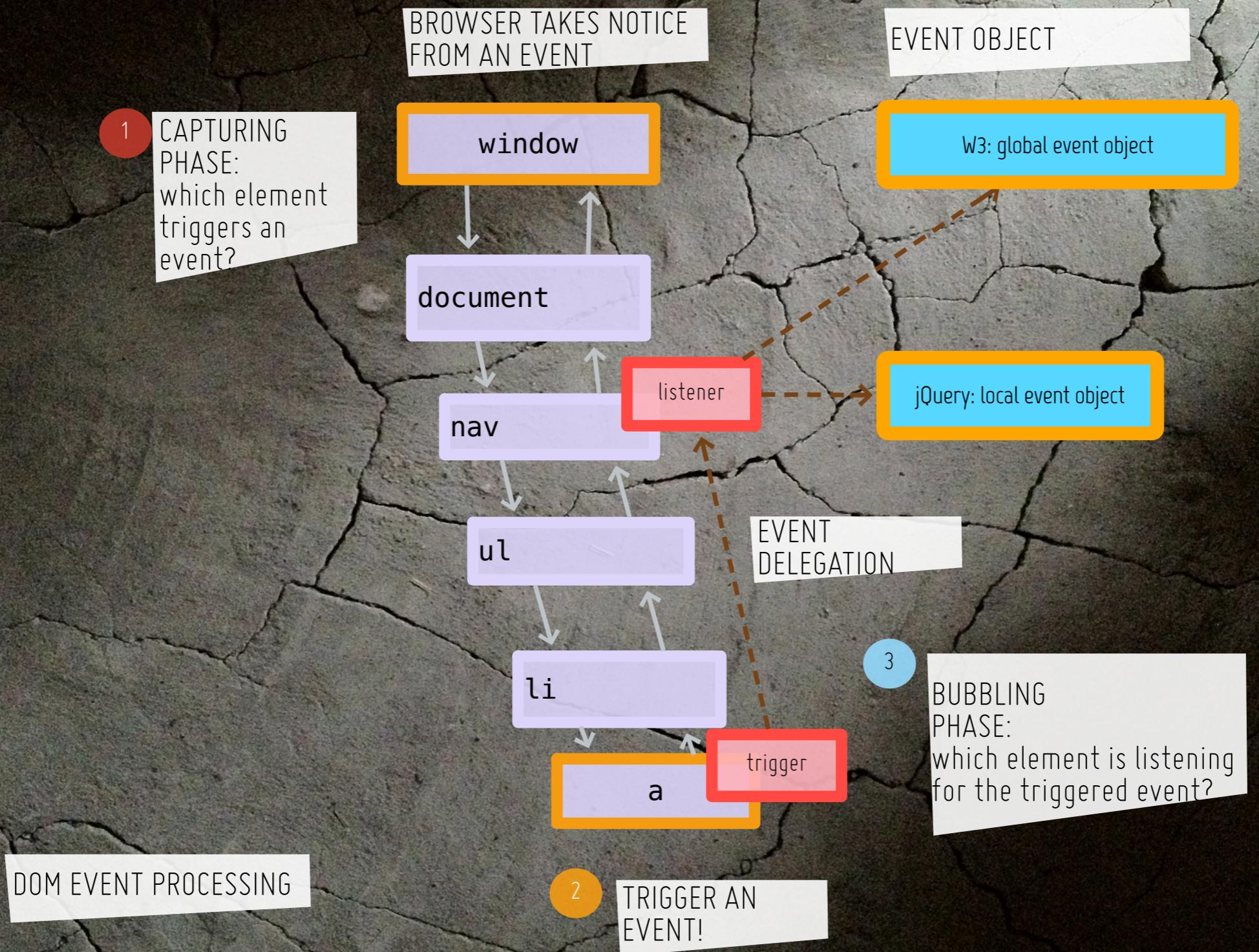
## **.empty()**

Remove all child nodes of the set of matched elements from the DOM.

## **.remove()**

Remove the set of matched elements from the DOM.

# EVENTVERARBEITUNG



## THE EVENT OBJECT

event  
target  
delegateTarget  
  
which  
type  
meta-/shift-/alt-/ctrlKey  
  
pageX  
pageY  
  
preventDefault()  
stopPropagation()  
  
...

## AN ONCLICK EVENT HANDLER

```
fn = {  
  onclick : function (event) {  
    event.preventDefault();  
    event.stopPropagation();  
    url = $(event.target).attr('href');  
    fn.loadContentOf(url)  
  },  
  loadContentOf : function (url) {...}  
}
```

USE `event.target`, NOT `this`!

# EVENTHANDLING

# DER ALLGEMEINE ADDEVENTLISTENER

## Ein Click Event

```
$("#box").addEventListener('click', function() {  
    alert('Etwas wurde geklickt');  
}, false);  
  
$("#box").addEventListener('click',  
    {name : value}  
    function(e) {  
        log(e.data.name);  
        alert('Etwas wurde geklickt');  
    }, false);  
}
```

# WEITERE SHORTHAND EVENTS FÜR MAUS UND TASTATUR

```
click()  
dblclick()  
mousedown()  
mouseenter()  
mouseleave()  
mousemove()  
mouseover()  
mouseout()  
hover()  
mouseup()
```

```
keydown()  
keypress()  
keyup()
```

```
$('form').keydown(function(e) {  
    if (e.which == '13' // Entertaste) {  
        e.preventDefault();  
    }  
})
```

# EVENTS FÜR FORMULARE

- .focus()** Erfasst, wenn ein DOM Element aktiv wird:  
Der Cursor betritt ein Inputelement.  
(siehe auch .focusin(), .focusout())
- .blur()** Der Cursor verlässt ein DOM Element, es wird inaktiv.
- .change()** Der Inhalt eines DOM Elementes ändert sich, der Wert eines Inputelementes wird überschrieben.
- .select()** Event beim Auswählen in einer Liste.
- .submit()** Event beim Abschicken eines Formulars.

# EVENTS FÜR DAS BROWSERFENSTER

- `resize()` Fenstergröße verändert sich,  
der User zieht das Fenster größer oder kleiner.
- `scroll()` Der User scrollt die Seite oder das Element.

# ON()

`.on( events [, selector] [, data], handler(eventObject) )`  
hängt einen oder mehrere Events (und namespaces), wie 'click'  
oder 'keydown.myPlugin' an eine Kollektion.

```
$("ul").on("click", doSomething);
```

```
$("ul").on("click tap myEvent", doSomething);
```

# `$(().on())` - EIN UNIVERSALER EVENTHANDLER

On wurde in jQuery 1.7 eingeführt und fasst die bisherigen Methoden zusammen:

`addEventListener()` Allgemeiner Eventlistener

`bind()/unbind()` Bindet und entfernt Events

`live ()/die()` Bindet und entfernt Events, auch an zukünftige DOM Objekte

`delegate()` Delegiert Events an Ancestors

# ON (SEIT 1.7)

```
.on( events [, selector] [, data], handler(eventObject) )
```

Der Handler wird bei Auslösen eines Events ausgeführt. Er erhält das Event-Objekt übergeben, das zum Beispiel mit `event.target` das getriggerte Objekt enthält, oder `event.data` übergebene Daten oder einiges mehr.

```
function doSomething(event) {  
  log(event.target); // The triggered object  
}  
  
$("ul").on("click", doSomething);
```

```
.on( events [, selector] [, data], handler(eventObject) )
```

Über das `data`-Objekt können zusätzlichen Daten in den Eventhandler übergeben werden. Data wird als `event.data` über das Eventobjekt in den Handler übergeben.

```
function doSomething(event) {  
  log(event.data.name); // Peter  
}  
  
$("ul").on("click", {name : "Peter"}, doSomething);
```

```
.on( events [, selector] [, data], handler(eventObject) )
```

Ein Selektor, der eine Delegation von Nachfahrenelementen ermöglicht. Er ersetzt.

```
 $("ul").on("click","a", doSomething);
```

# DELEGATION ERHÖHT DIE PERFORMANCE

Da jeder Eventlistener Speicherplatz und Prozesszyklen benötigt, ist das Delegieren von Events an ein Anchestorobjekt eine wichtige Methode, eine Applikation performant zu halten.

```
$("#dataTable tbody tr").on("click", function(event){  
    doSomething;  
});
```

Das tr Element wird "angewiesen", seinen empfangenen Event an das tbody hochzureichen. Dieses "Bubbling" genannte Verfahren ist Standard in Browsern, jQuery macht es sich hier zunutze.

```
$("#dataTable tbody").on("click", "tr", function(event){  
    alert($(this).text());  
});
```

# DAS EVENT OBJEKT

Im Event Objekt sind einige wichtige Eigenschaften und Methoden enthalten.

Zum Beispiel:

```
event.target  
event.delegateTarget  
event.currentTarget  
event.preventDefault()  
event.stopPropagation()
```

# STOP PROPAGATION()

Das Event wird daran gehindert, den DOM Baum nach oben zu wandern. Damit wird vermieden, dass Anchestor Elemente den Event "bemerken" und weiterverarbeiten.

# OFF, ONE

`$(sel).off()` schaltet Events wieder ab.  
`$("nav ul").on("click", "a", doSomething);`  
`$("nav ul").off("click", "a", doSomething);`

`$(sel).one()` schaltet einen Event nur einmal auf,  
nach dem Ausführen löscht sich der  
Event von selbst.

`$("nav ul").one("click", "a", doSomething);`

# .TRIGGER()

```
onMyEvent = function () { . . . }

$('sel').one('myEventFuerLadeDaten', onMyEvent);

$.trigger('myEventFuerLadeDaten');
```

Schreiben Sie eine Funktion, die das Abschicken des Formulars vor dem Browser abfängt und verarbeitet.

1. Welches Objekt liefert den Event "Abschicken"? -> `<form>`
2. Setzen des Eventlistener auf das `<form>` Element
3. Wie heisst der Event des Formulars? -> `submit`
4. Binden eines Eventhandlers `onSubmit` an den Eventlistener.
5. Verhindern der Browserverarbeitung
6. Ausgeben einer Konsolennachricht "`SUBMIT!`"

## AUFGABE

# MASSE UND POSITIONEN

# MASSE ERMITTELN

## **.height()**

Get the current computed height for the first element in the set of matched elements or set the height of every matched element.

## **.width()**

Get the current computed width for the first element in the set of matched elements or set the width of every matched element.

## **.innerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding but not border.

## **.innerWidth()**

Get the current computed inner width (including padding but not border) for the first element in the set of matched elements or set the inner width of every matched element.

# MASSE ERMITTELN

## **.innerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding but not border.

## **.innerWidth()**

Get the current computed inner width (including padding but not border) for the first element in the set of matched elements or set the inner width of every matched element.

## **.outerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns a number (without “px”) representation of the value or null if called on an empty set of elements.

## **.outerWidth()**

Get the current computed width for the first element in the set of matched elements, including padding and border.

# POSITIONEN ERMITTeln

## **.offset()**

Get the current coordinates of the first element, or set the coordinates of every element, in the set of matched elements, relative to the document.

## **.position()**

Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

# SCROLLEN

## **.scrollLeft()**

Get the current horizontal position of the scroll bar for the first element in the set of matched elements or set the horizontal position of the scroll bar for every matched element.

## **.scrollTop()**

Get the current vertical position of the scroll bar for the first element in the set of matched elements or set the vertical position of the scroll bar for every matched element.

# EFFEKTE - ZEIGEN UND VERSTECKEN

`.fadeIn(1000)`

Zeigt Elemente an, indem es allmählich die Deckkraft von 0 auf 1 einstellt. Dabei kann eine Zeit des Einblendens eingestellt werden.

`slow | normal | fast | millisekunden`

`.fadeOut()`

Blendet Elemente aus, indem es sie transparent macht,

## .fadeTo()

Zeigt Elemente an, indem es allmählich die Deckkraft von 0 auf einen Zielwert einstellt. Dabei kann eine Zeit des Einblendens eingestellt werden.

`.fadeToggle()`

Blendet Elemente abwechselnd ein und aus.

- `show(„slow“)`

Blendet ein Element über die Abmessungen und die Opacity ein.  
Zu Beginn wird das Element auf `display: block` gesetzt.

# .HIDE()

Blendet ein Element über die Abmessungen und die Opacity aus.  
Am Ende wird das Element auf display: none gesetzt.

```
hide(); // stellt unmittelbar auf display : none;  
  
// Ausblenden über Eigenschaften bis display: none;  
hide('slow' || 'normal' || 'fast');  
  
hide(1000); // Ausblenden innerhalb 1000 Millisekunden
```

`.toggle()`

Zeigt oder versteckt die ausgewählten Elemente.

## `.slideDown()`

Blendet ein Element ein, indem es nach unten 'aufklappt'.

## `.slideUp()`

Blendet ein Element wieder aus, indem es nach oben 'zuklappt'.

`.slideToggle()`

'Klappt' ein Element auf und zu.

`.stop()`

Hält eine aktuell laufende Animation an.

`stop(true, true);`

1. `true` → stoppt alle wartenden  
Animationen und löscht  
sie aus der Queue

2. `true` → stoppt die laufende Animation

## .delay()

Bei Verkettung von mehreren Animationen kann mit delay eine Pause zwischen den Animationen eingestellt werden.

```
$('box')
  .slideDown(1000)
  .delay(2000)
  .slideUp(1000)
```

## **• queue( )**

Zeigt die Queue von Funktionen an, die momentan auf der aktuellen Collection ausgeführt werden.

## `.clearQueue()`

entfernt alle Aktionen aus der Queue, die noch nicht ausgeführt worden sind.

`.dequeue()`

Führt die nächste Aktion der Queue des aktuellen Elements aus.

# .ANIMATE()

Ermöglicht die gleichzeitige Animation verschiedener CSS-Eigenschaften.

```
$('p')
.animate( {
    height: '300px',
    width:  '600px',
    font-size : '36px'
} )
```

# .ANIMATE()

Dabei können die Eigenschaften, eine Dauer, der Easing-Effekt, sowie eine Callback-Funktion angegeben werden.

```
$('p')
  .animate(
    { left: '100px' },
    { duration: 'slow' },
    'swing'
  )
```

# .ANIMATE()

Dabei können die Eigenschaften, eine Dauer, der Easing-Effekt, sowie eine Callback-Funktion angegeben werden.

```
$('p')
  .animate(
    { left: '100px' },
    1000,
    'swing'
  )
```

# Bringen Sie den folgenden jQuery Code für einen Image Slider in eine anständige Form.

1. Sortieren Sie die Codezeilen nach Declaration, Methods und Controls.
2. Dokumentieren Sie dort, wo notwendig.
3. Deklarieren Sie Variablen und Methoden in einer Single-var Anweisung
4. Benennen Sie die Methoden um nach `fn = function () {}`
5. Tauschen Sie alle Convenience Listener gegen `$(()).on()` aus.
6. Sammeln sie die Controls in einer `main()` Methode und schreiben sie ein Main Control.

[HTTPS://CODEPEN.IO/DOODLEMARKS/PEN/AFCLY](https://codepen.io/doodlemarks/pen/afcly)

# TRAVERSIEREN

# VERKETTUNG UND TRAVERSIEREN

```
<script type="text/javascript">
$(document).ready(function() {
    var obj = $("#box p");
    obj.click(function() {
        var pos = obj.index(this);
        obj.removeClass().parent().removeClass();
        $(this).addClass("green")
            .parent()
            .addClass
            ("boxColor-" + pos);
    });
});
</script>
```

# TRAVERSIEREN IM DOM

Beim Traversieren wird die aktuelle Collection verändert. Ihr werden entweder Elemente hinzugefügt, oder es findet eine Veränderung der Collection durch Bewegung auf einer der Dokumentenachsen statt.

Dies ist dann erforderlich, wenn der zu selektierende Knoten nicht auf direktem Weg erreicht werden kann.

# NOCHMAL DIE ACHSEN IM DOKUMENT

Das Bewegen auf den Dokumentenachsen:

Sibling – Achse

Gleiche hierarchische Stufe zu den Geschwisterelementen

Child-/Descendant – Achse

Zu den Kindknoten

Parent-/Ancestor – Achse

Zu den Elternknoten

# .CHILDREN()

Erfasst die Kindelemente jedes Elementes in der Collection.  
Optional kann ein Filter eingesetzt werden. Erfasst keine  
Textknoten.

```
$('#box1')
  .css({border:'1px solid green',padding:'15px'})
  .children('p:nth-child(2n)').css({border:'1px solid red'})
```

# .CHILDREN()

```
$('#box1')
  .css({border:'1px solid green',padding:'15px'})
  .children().css({border:'1px solid red'})
```

...

```
<div id="box1">Dies ist Box 1.
  <p id="p1">Der erste Textabsatz in div#box1.</p>
  <p id="p2">Der zweite Textabsatz in div#box1.</p>
  <div>Dies ist ein Div-Container in div#box1.</div>
</div>
```

# .CONTENTS()

Erstellt eine Collection aus allen Kindknoten eines Elements, inklusive der Textknoten und leeren Textknoten (Zeilenumbrüche).

```
alert($('#box1').contents().length)
```

```
<div id="box1">Dies ist Box 1.  
  <p id="p1">Der erste Textabsatz in div#box1.</p>  
  <p id="p2">Der zweite Textabsatz in div#box1.</p>  
  <div>Dies ist ein Div-Container in div#box1.</div>  
</div>
```

# .PARENT()

Ermittelt das Elternelement zu jedem Element der Collection.

```
$('>#box>p, li')  
  .css('border':'1px solid red')  
  .parent()  
  .css('border':'1px solid green');
```

```
<div id="box1">Dies ist Box 1.  
  <p id="p1">Der erste Textabsatz in div#box1.</p>  
  <p id="p2">Der zweite Textabsatz in div#box1.</p>  
  <div>Dies ist ein Div-Container in div#box1.</div>  
</div>  
<ul>  
  <li>Listenpunkt</li>  
</ul>
```

# .PARENTS()

Erzeugt eine neue Collection aus allen Elementen der Ancestor-Achse aller Elemente der aktuellen Collection.  
Optional kann ein Selector verwendet werden.

```
$(document).ready(function() {
  $('#p3')
    .css({border:'1px solid red'})
    .parents('div')
    .css({border:'1px solid green'})
});
```

```
<div id="box1">Dies ist Box 1.
  <p id="p1">Der erste Textabsatz in div#box1.</p>
  <p id="p2">Der zweite Textabsatz in div#box1.</p>
  <div id="d1"><p id="p3">container in box1</p></div>
</div>
```

# .PARENTSUNTIL()

Erzeugt eine neue Collection aus allen Elementen der Anchestor-Achse aller Elemente der aktuellen Collection bis zu dem dem Selektor entsprechenden Element.

Letzteres wird nicht mit in die Collection aufgenommen.

```
$(document).ready(function() {  
  $('#p3')  
    .css({border:'1px solid red'})  
    .parentsUntil('div')  
    .css({border:'1px solid green'})  
});
```

# .SIBLINGS()

Erfasst alle Geschwisterelemente des ausgewählten Selektors.

```
$('p')
  .siblings()
  .css('border':'1px solid green')
```

# .NEXT()

```
$('#p1')
  .css({border:'1px solid red'})
  .next()
  .css({border:'1px solid green'})  
...  
<div id="box1">Dies ist Box 1.  
  <p id="p1">Der erste Textabsatz in div#box1.</p>  
  <p id="p2">Der zweite Textabsatz in div#box1.</p>  
  <div>Dies ist ein Div-Container in div#box1.</div>
</div>
```

# .NEXTALL()

Erfasst alle der Collection folgenden Sibling-Elemente. Es kann auch hier ein Selektor verwendet werden, der Suche nach dem nächsten Element entsprechend spezifiziert.

```
$('#p1')
  .css({border:'1px solid red'})
  .nextAll()
  .css({border:'1px solid green'})
```

```
<div id="box1">Dies ist Box 1.
  <p id="p1">Der erste Textabsatz in div#box1.</p>
  <p id="p2">Der zweite Textabsatz in div#box1.</p>
  <div><p>Dies ist ein Div-Container in box1.</p></div>
</div>
```

# .NEXTUNTIL()

Erfasst alle der Collection folgenden Sibling-Elemente bis zum Element, das dem Selektor entspricht. Das Selektorelement selbst ist nicht Bestandteil der Collection.

```
$('#p1')
  .css({border:'1px solid red'})
  .nextUntil('div')
  .css({border:'1px solid green'})
```

# .prev()

Erfasst das umittelbar der Collection vorausgehende Element, optional durch einen Selektor genauer bestimmbar, und bildet eine neue Collection.

```
$('p2')
  .css('border':'1px solid red')
  .prev()
  .css('border':'1px solid green')
```

# .prevAll()

Erfasst alle vorausgegangenen Geschwisterknoten und bildet eine neue Collection. Mit einem Selector können die Elemente eingeschränkt werden.

```
$('p2')  
  .css('border':'1px solid red')  
  .prevAll()  
  .css('border':'1px solid green')
```

# .PREVUNTIL()

Erfasst alle vorausgegangenen Geschwisterknoten bis zu dem dem Selektor entsprechenden Element und bildet eine neue Collection. Mit einem Selector können die Elemente eingeschränkt werden.

# .CLOSEST()

Erfasst den der Selection am nächsten liegenden Knoten im Dom auf der Ascendant–Achse, der der Selection entspricht. Dies kann der Parent sein, aber auch ein anderer Knoten.

```
$('#p3')
  .css({border:'1px solid red'})
  .closest('div')
  .css({border:'1px solid green'})
```

```
<div id="box1">Dies ist Box 1.
  <p id="p1">Der erste Textabsatz in div#box1.</p>
  <p id="p2">Der zweite Textabsatz in div#box1.</p>
  <div id="d1"><p id="p3">container in box1</p></div>
</div>
```

# .FIND()

Selektiert auf der Descendant-Achse in beliebiger Hierarchietiefe, ausgehend von der aktuellen Collection. In der Regel wird find() mit einem Selectorelement verwandt.

```
$('#box1')
  .css({border:'1px solid red'})
  .find('p')
  .css({border:'1px solid green'})
```

```
<div id="box1">Dies ist Box 1.
  <p id="p1">Der erste Textabsatz in div#box1.</p>
  <p id="p2">Der zweite Textabsatz in div#box1.</p>
  <div id="d1">
    <p id="p3">Dies ist ein p-Container</p>
  </div>
</div>
```

Wie lässt sich der folgende Code mit Hilfe von  
jQuery-Traversen "in einem Stück" herstellen vor  
dem Button einfügen?

```
<div id="keep">  
  <input type="checkbox" id="cb">  
  <label for="cb">keep Password</label>  
</div>
```

AUFGABE

# COLLECTIONS ERWEITERN

# .ADD()

Erweitert die aktuelle Collection um weitere Elemente.

...

# .ADD()

```
$(document).ready(function() {
    var absatz1 = document.getElementById('p1');
    var absatz2 = document.getElementById('p2');

    $('#p1').click( function() {
        $('#p1').add('#p2').css({color:'red'})
    });
});

...
<div id="box1">
    <p id="p1">Klick für Add-Test</p>
    <p id="p2">Beide Absätze färben sich rot.</p>
</div>
```

# .ANDSELF()

Beim Traversieren verschiebt sich die Collection auf die Elemente, auf die sich die Traversierung richtet. Um nach der Traverse die ursprüngliche Collection zu behalten, wird der Befehl "andSelf()" angewandt.

```
$("#box").children('p').andSelf();
```

# .ANDSELF()

```
$(document).ready(function() {  
    $('#box1')  
        .children('p')  
        .andSelf()  
        .css({border:'1px solid green'})  
});  
  
...  
  
<div id="box1">Dies ist Box 1.  
    <p id="p1">Der erste Textabsatz in div#box1.</p>  
    <p id="p2">Der zweite Textabsatz in div#box1.</p>  
    <div id="d1"><p id="p3">Dies ist ein Div-Container in  
div#box1, einen Textabsatz enthält.</p></div>  
</div>
```

# COLLECTIONS FILTERN

## .EQ()

Reduziert die aktuelle Collection auf das Element mit dem spezifizierten Index.

```
$('#box p')
.eq(1).text('Nur dieses Element wird geändert.')
```

# .FILTER()

Reduziert eine Collection auf die dem Filtern entsprechenden Elemente. Filter ist ein beliebiger Selektor.

```
$('p')
  .filter(':contains(zweite)')
  .css('border':'1px solid green')
```

```
$('p')
  .filter(
    function () {
      return ($(this).parents('div').length < 2);
    }
  )
  .css('border':'1px solid green')
```

# .FIRST()

Reduziert die Collection auf deren erstes Element.

```
$('p')
  .first()
  .css({border:'1px solid green'})
```

...

```
<div id="box1">Dies ist Box 1.
  <p id="p1">Der erste Textabsatz in div#box1.</p>
  <p id="p2">Der zweite Textabsatz in div#box1.</p>
  <div><p id="p3">Dies ist ein Div-Container in div#box1,
  einen Textabsatz enthält.</p></div>
</div>
```

# .HAS()

Selektiert Elemente aus einer Collection, die wenigstens ein Element enthalten, das dem Filter entspricht.

```
$('div')
  .has('div')
  .css({border:'1px solid green'})
```

# .LAST()

Reduziert die Collection auf deren letztes Element.

```
$('p')
  .last()
  css({border:'1px solid green'})
```

...

```
<div id="box1">Dies ist Box 1.
  <p id="p1">Der erste Textabsatz in div#box1.</p>
  <p id="p2">Der zweite Textabsatz in div#box1.</p>
  <div><p id="p3">Dies ist ein Div-Container in div#box1,
  einen Textabsatz enthält.</p></div>
</div>
```

# .NOT()

entfernt aus einer Collection alle Elemente, die dem not-Filter entsprechen. Dies entspricht einem negativen Filter.

```
$('p')
  .not(':contains(zweite)')
  .css('border':'1px solid green')
```

```
$('p')
  .not(
    function (
      return ($(this).parents('div').length < 2);
    )
  .css('border':'1px solid green')
```

# .SLICE()

Reduziert die aktuelle Collection auf die im Slice-Index eingegrenzte Menge. Bei Angabe nur eines Indexwertes erhält die Collection alle Elemente ab dem angegebenen Indexwert.

```
$('p')
  .slice(1,3)
  .css('border':'1px solid green')
```

```
$('p')
  .slice(1)
  .css('border':'1px solid green')
```

# WEITERE FUNKTIONEN

# .EACH()

Führt eine Funktion für jedes Element einer Collection aus.  
Dabei erhält die Funktion den Index aus dem Iterationsloop und  
ggf. das aktuelle Element daraus.

```
each(fn(i, elem))

$.each( $('#box p'),
        function(index, item) {
            var inc = i+1;
            $(this).text ('Das ist
                          der ' + inc + ' . Absatz');
        });

```

# .END()

Hebt alle aktuellen Filterungen auf und stellt die ursprüngliche Collection wieder her.

```
$('p')
  .first().css('border':'1px solid green')
  .end()
  .last().css('border':'1px solid red')
  .end()
  .css('font-style':'italic');
```

.END()

```
<div id="box1">Dies ist Box 1.  
  <p id="p1">Der erste Textabsatz in div#box1.</p>  
  <p id="p2">Der zweite Textabsatz in div#box1.</p>  
  <div><p id="p3">Dies ist ein Div-Container in div#box1,  
  einen Textabsatz enthält.</p></div>  
</div>
```

# .IS()

Gibt ein boolsches true oder false zurück, jenachdem ob der Knoten aus der aktuellen Collection dem Filtermerkmal entspricht oder nicht.

```
$('div').each(  
  function(){  
    if(this).is('#box'){ // === $('div#box')  
      $(this).text('hier ist die Box.');//  
    } else {  
      $(this).text('hier ist sie nicht.');//  
    }  
  }  
)
```

# .CONTEXT

Gibt ein Kontextobjekt zurück

Gibt den Elternknoten zurück, falls der an `$()` übergeben wurde. Sonst das document-object.

# GET(N)

gibt true oder false zurück, je nach Selektor

Erstellt ein Array aus den DOM–Elementen der aktuellen Collection.

Oder gibt den Elementknoten des Items der Collections zurück, das sich an der Indexposition n befindet.

```
var images = $("#box img").get(i);  
alert(images.src)
```

# INDEX(SEL)

Gibt den Index des ersten Elements in der index(elem) aktuellen Collection zurück, das durch sel oder elem beschrieben wird.

Ohne Angabe des Index wird die Position des ersten Items der Collection unter den Kindknoten seines Parents zurück gegeben.

# IS(SEL)

gibt true oder false zurück, je nach Selektor.

# .LENGTH

Liefert die Anzahl der Elemente der Collection

# .SIZE

Liefert die Anzahl der DOM Elemente der Collection

# .TOARRAY

Liefert ein Array aus den Elementen der aktuellen Collection

# .OFFSETPARENT()

Erfasst das nächstliegende Anchestorelement oder den Bezugscontainer zurück, der bei der Positionierung des aktuellen Elementes aus der Collection relevant ist.

```
div { left: 10; width:100px; }

<div>
  <p><b>Text mit Auszeichnung</b> </p>
</div>

var offsetObj = $('b').offsetParent;
log(offsetObj.x);
```

# A J A X

# ASYNCHRONOUS JAVASCRIPT AND XML

- AJAX ist eine Methode, mittels der ein Browser asynchrone Requests an den Server stellen kann. Grundlage bildet das XMLHttpRequest Objekt, das in der Javascript Version 1.5. hinzugekommen ist.
- Das Akronym AJAX geht vermutlich auf Jesse James Garrett (Mitarbeiter der Agentur Adaptive Path) in seinem Aufsatz *Ajax: A New Approach to Web Applications* vom 18. Februar 2005 zurück.

# XMLHTTPREQUEST

- Das XMLHttpRequestObjekt wurde 2004 von Microsoft entwickelt (oder Mitentwickelt), später wurde es in den Javascript Standard aufgenommen.
- Daher haben lange zeit zwei verschiedene Implementationen für den IE und die anderen Browser koexistiert.

```
//Microsoft:  
xmlhttp = new ActiveXObject("Microsoft.XMLHTTP"); //IE > 5  
xmlhttp = new ActiveXObject("Msxml2.XMLHTTP"); //IE > 6  
  
//Standardbrowser  
xmlhttp = new XMLHttpRequest(); // Standard und IE > 7
```

# READYSTATE ONREADYSTATECHANGE

```
// Während eines Request wechseln die readystate - Werte  
// onreadystatechange erfasst die Änderungen.  
  
0 Uninitialized – The initial value.  
1 Open – The open() method has been successfully called.  
2 Sent – The UA successfully completed the request,  
        but no data has yet been received.  
3 Receiving – Immediately before receiving the message body  
        (if any). All HTTP headers have been received.  
4 Loaded – The data transfer has been completed.
```

# AJAX PER JAVASCRIPT

```
var xmlhttp = null;
try {
    // Mozilla, Opera, Safari sowie Internet Explorer (ab v7)
    xmlhttp = new XMLHttpRequest();
} catch(e) {
    try {
        // MS Internet Explorer (ab v6)
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    } catch(e) {
        try {
            // MS Internet Explorer (ab v5)
            xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
        } catch(e) {
            xmlhttp = null;
        }
    }
}
if (xmlhttp) {
    xmlhttp.open('GET', 'beispiel.xml', true);
    xmlhttp.onreadystatechange = function () {
        if (xmlhttp.readyState === 4) {
            alert(xmlhttp.responseText);
            xmlhttp.send(null);
        }
    };
    doSomethingElse();
}
```

# AJAX UND JQUERY

- jQuery besitzt eine eigene Implementation von AJAX-Methoden, die XMLHttpRequest Handhabung vereinfachen.
- Darin finden sich Methoden wie ajax(), get(), post() und weitere.

# .AJAX() - IST DIE BASISFUNKTION FÜR EINEN AJAX REQUEST.

```
$("#trigger").click(function(){
  $.ajax({
    type      : "GET" || "POST",
    url       : "ajax.php",
    context   : $(node),
    data      : daten,
    dataType  : "html || json || jsonp|| xml || script",
    cache     : false,
    async     : true,
    timeout   : 2000,
    statusCode : {
      404: function() {
        alert("page not found")
      }
    },
    success   : function(data){ $(this)..... },
    error     : function(jqxhr, status, errorThrown) {}
  });
});
```

# AJAX USES PROMISES!

```
$("#trigger").click(function(){
    request = $.ajax({
        type      : "GET" || "POST",
        url       : "ajax.php",
        context   : $(node),
        data      : daten,
        dataType  : "html || json || jsonp|| xml || script",
        cache     : false,
        async     : true,
        timeout   : 2000,
        statusCode : {
            404: function() {
                alert("page not found");
            }
        }
    });
    request
        .done(function(response) { ... })
        .fail(function(jqxhr, status, errorThrown) { ... });
});
```

# .AJAXCOMPLETE()

- `ajax({ complete : function(){ ... } })`
- Erstellt und registriert einen Handler, der aufgerufen wird, wenn ein AJAX Request erfolgreich beendet wurde.

# .AJAXERROR()

- `ajax({ error : function(jqXHR, textStatus, errorThrown){ ... } })`
- Handler, der aufgerufen wird, wenn ein AJAX Request mit einem Fehler beendet wird. Dabei werden ein Fehlerobject und zwei Fehlermeldungen übergeben.

# \$.AJAXSUCCESS()

- `ajax({ success : function(){ ... } })`
- Fügt eine Funktion hinzu, die immer dann ausgeführt wird, wenn ein AJAX Event erfolgreich beendet wurde.

```
ajax({ success : function(data){  
    workWith(data);  
    trigger("fireMyEvent");  
}  
})  
$('obj').addEventListener('fireMyEvent', function()  
{$.ajax() ...});
```

# .AJAXSEND()

- `ajax({ beforeSend : function(jqXHR, settings){ ... } })`
- Fügt eine Funktion hinzu, die ausgeführt wird, bevor ein AJAX Request abgeschickt wird.

AjaxSend wird als Ajax Event ausgeführt.

```
var formData;
$.ajax({
  beforeSend : function(jqXHR, settings){
    formData = $('.myForm').serialize();
  },
  data : formData,
  success : function ...
})
```

# .AJAXSTART()

- Erstellt und registriert einen Handler, der aufgerufen wird, wenn ein AJAX Request startet.

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>

// We can attach our event handler to any element:
$('.log').ajaxStart(function() {
  $(this).text('Triggered ajaxStart handler.');
});

// Now, we can make an Ajax request using any jQuery method:
$('.trigger').click(function() {
  $('.result').load('ajax/test.html');
});
```

# .AJAXSTOP()

- Erstellt und registriert einen Handler, der aufgerufen wird, wenn ein AJAX Requests beendet wurde.

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>

// We can attach our event handler to any element:
$('.log').ajaxStop(function() {
  $(this).text('Triggered ajaxStop handler.');
});

// Now, we can make an Ajax request using any jQuery method:
$('.trigger').click(function() {
  $('.result').load('ajax/test.html');
});
```

# JQUERY.AJAXSETUP()

- Mit ajaxSetup() werden die Einstellungen aller (zukünftigen) AJAX Requests eingestellt.
- Alle nach den Setup auszuführenden ajax() - Requests richten sich nach diesen Angaben.

```
$.ajaxSetup({
    type      : "GET" || "POST",
    url       : "ajax.php",
    context   : $(node),
    dataType : "html || json || xml",
    cache    : false,
    async    : true
});
```

# SHORT HAND METHODS

```
{  
    "blume1" : { "Name" : "Rose", "Farbe" : "rot" },  
    "blume2" : { "Name" : "Tulpe", "Farbe" : "gelb" },  
    "blume3" : { "Name" : "Nelke", "Farbe" : "blau" }  
}  
  
$.getJSON('blumen.json', function(result){  
    $.each(result, function(index, value){  
        $('#output').append('<li>' + value['Name'] + '</li>')  
    })  
})
```

JSONP (JSON WITH PADDING) REQUESTS FALLEN  
NICHT UNTER DIE EINSCHRÄNKUNGEN DER SAME  
ORIGIN POLICY RESTRICTIONS.

```
$.getJSON("http://api.flickr.com/services/feeds/  
photos_public.gne?jsoncallback?",  
{  
    tags      : "cat",  
    tagmode   : "any",  
    format    : "json"  
},  
function(data) {  
    $.each(data.items, function(i,item){  
        $("<img/>").attr("src",  
item.media.m).appendTo("#images");  
        if ( i === 3 ) return false;  
    });  
});
```

# .GET(), .POST()

```
$ .get('ajax/test.html', function(data) {  
    $('.result').html(data);  
    alert('Load was performed.');//  
});  
  
$("#trigger").click(function(){  
    $.post("ajax.php",  
    {  
        ajaxpost: "post()-Daten"  
    },  
    function(data){  
        $("#content").html(data);  
    } );  
});
```

# .GETSCRIPT() LIEST UND KOMPILIERT!

```
$.getScript("ajax/test.js", function(data, textStatus, jqxhr)
{
    console.log(data); //data returned
    console.log(textStatus); //success
    console.log(jqxhr.status); //200
    console.log('Load was performed.');
});

// Fehlerbehandlung
$.getScript("ajax/test.js")
.done(function(script, textStatus) {
    console.log( textStatus );
})
.fail(function(jqxhr, settings, exception) {
    $( "div.log" ).text( "Triggered ajaxError handler." );
});
```

```
$('#BOX').LOAD('PAGE.HTML #CONTENT1')
```

Lädt Daten vom Server und platziert diese anschließend als HTML in das selektierte Element.

**page.html:**

```
<div id="content1">...</div>
<div id="content2">...</div>
```

# DATEN MIT DEM REQUEST MITSCHICKEN

```
request = $.ajax({
  type : 'get', // 'post'
  url  : 'server.php',
  data : {key1 : 'value', key2 : 'value'},
  ...
})
```

```
$( '#MYFORM' ).SERIALIZE()
```

- Wandelt die Daten eines Formulars in einen (serialisierten) String um, der per Request an den Server übermittelt werden kann.

```
$ajax({  
    url : $('form#login').attr('action'),  
    data : $('form#login').serialize(),  
    ...  
})
```

```
$( '#MYFORM' ).SERIALIZEARRAY()
```

- Wandelt die Daten eines Formulars in ein Array um, das weiter bearbeitet werden kann.

```
var array = $('form#login').serializeArray();
for (var i=0; i<array.length; i++) {
    validate(array[i]);
}

function validate (value) { ... }
```

# MEHRERE AJAX REQUESTS ?

# REQUESTS ASYNCHRON ODER SYNCHRON STEUERN

- Standardmäßig werden Ajax Requests asynchron ausgeführt (async : true). Jeder Request wird unabhängig von anderen ausgeführt. jQuery verwendet für jeden Request ein eigenes XHR Objekt.

- Asynchronizität bedeutet aber auch, dass jeder Request liefert, wenn er fertig ist. Nicht unbedingt in der Reihenfolge des Abschickens.

```
$.AJAX({ASYNC : FALSE});
```

- Sollen Ajax Request hintereinander ausgeführt werden, so muss das entsprechend eingestellt sein
- Dann wird ein Ajax Request erst dann ausgeführt, wenn ein vorhergehender zu Ende ist. Dies blockiert unter Umständen den Browser solange, bis alle Requests ausgeführt sind.
- Crossdomain Requests lassen einen synchronen Request nicht zu.

# AJAX CALLBACK KETTEN

# AJAX REQUESTS ALS CALLBACK-KETTE

```
//Erster Request:  
request[0] = $ajax({ ... });  
request[0].done(function (response) {  
  
    // Zweiter Request  
    request[1] = $ajax({ ... });  
    request[1].done(function (response) {  
  
    })  
})
```

ORIGINAL VON BY JOSHUA FLANAGAN, 20. 10. 2011

MEHRERE AJAX REQUESTS  
MIT JQUERY.WHEN

# LOREM IPSUM DOLOR

Stellen Sie sich zum Beispiel vor, sie möchten Tweets von drei verschiedenen Usern einsammeln, um die Inhalte anschließend alphabetisch sortiert anzuzeigen. Um einen Tweet zu bekommen, schreiben Sie eine Funktion:

```
$.get("http://twitter.com/status/user_timeline/  
SCREEN_NAME.json",  
function(tweets){  
    // work with tweets  
},  
"jsonp");
```

- Um drei Tweets abzufragen, braucht es drei unabhängige `$.get` Aufrufe. Solange jeder Aufruf asynchron ausgeführt wird, gibt es keine Gewähr dafür, in welcher Reihenfolge die Ergebnisse geliefert werden.

```
$.$when( $.ajax("server.php") )
  .then(function.ajaxArgs){
    alert.ajaxArgs[1]);
    /* ajaxArgs is [ "success", statusText, jqXHR ] */
});
```

- Unsere Anwendung nimmt eine Liste von "Deferred Objects" auf und erzeugt einen Callbackhandler, der ausgeführt wird, wenn alle zurückgestellten Objekte vollständig abgearbeitet sind.
- Deferred Objects werden geliefert, wenn deren Ajax Funktion vollständig abgearbeitet sind (complete).

```
$.$when( getTweets('twitterUsername1'),  
        getTweets('twitterUsername2'),  
        getTweets('twitterUsername3') )  
    .done(function(tweet1Args, tweet2Args, tweet3Args){  
  
        var allTweets = []  
            .concat(tweet1Args[0])  
            .concat(tweet2Args[0])  
            .concat(tweet3Args[0]);  
  
        var sortedTweets = sortTweets(allTweets);  
        showTweets(sortedTweets);  
    });  
});  
  
// see it in action at http://jsfiddle.net/94PGy/4/
```