

Modernes JavaScript

Rüstzeug für Webapp-Entwicklung

0. jQuery

DSL für DOM



Merke: Wer es nicht nutzt, macht es falsch!

(Höchstwahrscheinlich)

jQuery

- Library für clientseitiges Scripting
- Entwicklung ab 2006 durch John Resig u.A., MIT-Lizenz; sehr populär
- Schwerpunkte:
 - DOM-Manipulations-Abstraktion
 - AJAX und Effekte
 - Diverse Helper-Funktionen

// Normales DOM-JavaScript

```
var tables = document.getElementsByTagName('table');
for(var t = 0; t < tables.length; t++){
    var rows = tables[t].getElementsByTagName('tr');
    for(var i = 0; i < rows.length; i += 2){
        if(!/^(s)odd(s)$/.test(rows[i].className)){
            rows[i].className += 'odd';
        }
    }
}
```

```
// Das gleiche mit jQuery  
$('table tr:nth-child(odd)')  
  .addClass('odd');  
  .text('Hallo Welt!');
```

// Flexible Methoden

// Man kann es so machen

```
$('nav').attr('id', 'foo').attr('lang', 'de');
```

// ... oder so:

```
$('nav').attr({  
  id: 'foo',  
  lang: 'de'  
});
```

// ... und mit nur einem Argument

// ist .attr() ein Getter!

// Getter und Setter

```
var text = $('.foo').text(); // Text auslesen  
$('.foo').text(text);      // Text setzen
```

```
var html = $('#Foo').html(); // HTML auslesen  
$('.bar').html(html);       // HTML schreiben
```



```
// Tipp: $() kann aus HTML-Input DOM erzeugen  
$( '<section>' ).text( 'Möp' ).appendTo( 'body' );
```

```
// Tipp: jQuery-Objekte cachen  
$content = $( '#Content > div' );
```

// Tipp: Verketten, verketten, verketten!

```
$( 'ul' )  
  .attr( 'id', 'nav' )  
  .find( 'li' )  
  .addClass( 'navListItem' )  
  .find( 'a' )  
  .each( function() {  
    $( this ).attr( 'href', '/' + $( this ).text() );  
  } );
```

// DOM-Navigation

next(), find(), closest(), parents(), children()

// Events

`$('form').on('submit', function(evt){ });`

// Multi-Events I

`$('.foo').on('mouseover mouseout', function(evt){ });`

// Multi-Events II

`$('.bar').on({
 mouseover: function(){ },
 mouseout: function(){ }
});`

// Ajax

`$.get('/api/foo', function(result){ });`

// Click-Event

```
$('#Foo').on('click', function(evt){  
    window.alert('Hallo Welt!');  
});
```

// Mehrere Events

```
$('#Foo').on('mouseover mouseout', function(evt){  
    window.alert('Hallo Welt!');  
});
```

// Event delegation

```
$('#Foo').on('mouseover mouseout', '.bar', function(evt){  
    window.alert('Hallo Welt!');  
});
```

Fazit jQuery

- **jQuery = DSL für das DOM**
- Browserbugs wegabstrahiert, Features nachgerüstet
- `$('#Foo').foo().bar().baz()`, fertig!
- **Wer es nicht nutzt, macht es falsch!**
- Dokumentation: api.jquery.com

Alles klar zu jQuery?

1. AMD

Frontend-Module für JavaScript

JS hat kein Modulsystem *

```
<script src="jquery.js"></script>  
<script src="underscore.js"></script>  
<script src="backbone.js"></script>  
<script src="coffee-script.js"></script>  
<script src="app.js"></script>
```

- Keine Dependencies
- Sequenzieller Download
- Globale Variablen

Asynchronous Module Definition

- AMD = **Community-Standard für Frontend-Module**
- Nicht „offiziell“, aber weit verbreitet – Umsetzung per JS-Lib(s)
- Definiert zwei API-Funktionen und ein paar Konfigurations-Optionen


```
define('name', ['d1', 'd2'], callback);
```

1. `define()` wird durch AMD bereitgestellt
2. Modulname ist optional
3. Abhängigkeiten werden geladen – können selbst AMD-Module sein
4. Der Callback mit dem Modul-Code feuert, wenn alle Abhängigkeiten geladen wurden

Modul-APIs

Der Rückgabewert des Callbacks in einem `define()` ist die Modul-API!

```
// antwort.js  
define(function(){  
    return 42;  
});
```

Modul-APIs verwenden

Modul-APIs von Abhängigkeiten werden als Arguments von Modul-Callbacks übergeben

```
// frage.js  
define(['antwort'], function(antwort){  
    alert(antwort); // 42  
});
```

Non-AMD-Scripts

Scripts, die kein AMD-Modul sind, geben nichts zurück, funktionieren aber weiterhin wie gewohnt

```
define(['keinAmd'], function(foo){  
    alert(typeof foo);           // "undefined"  
    alert(typeof window.foo);    // "object"  
});
```

require()

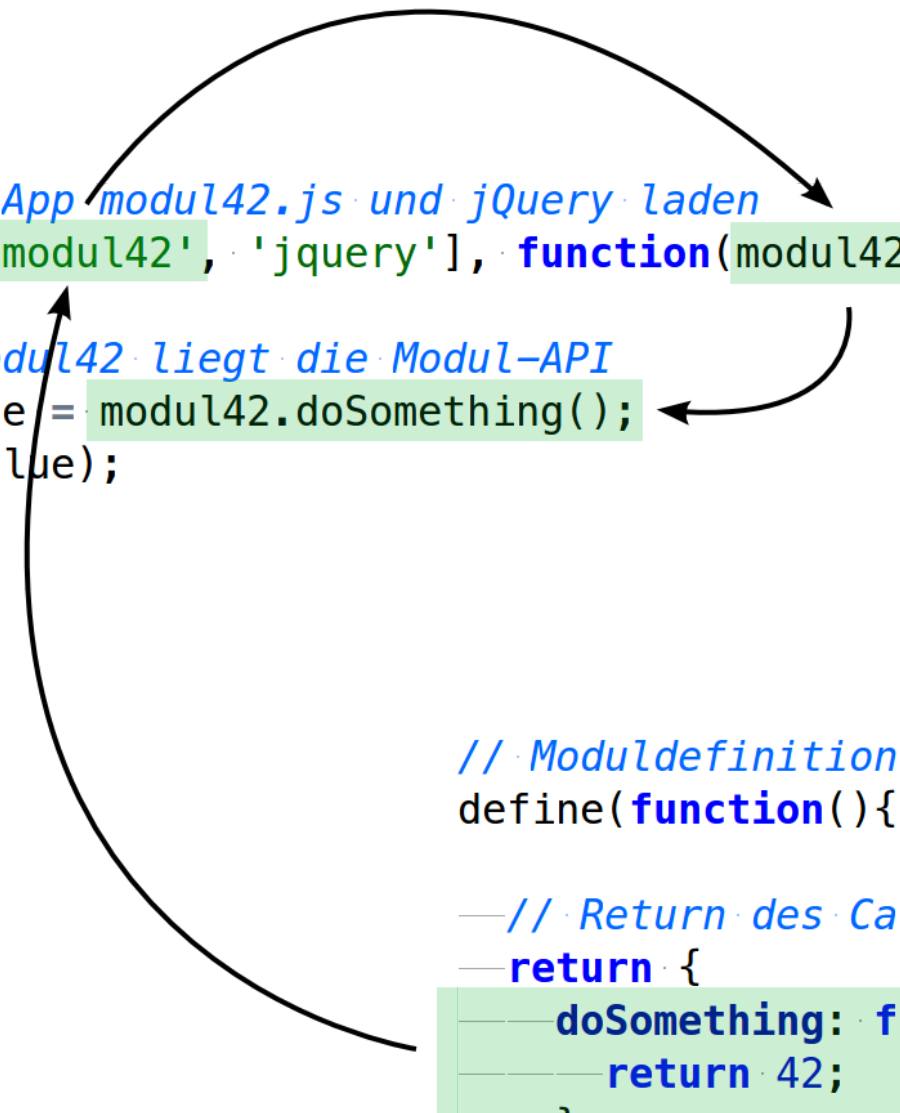
Lädt Module, ohne selbst eines zu definieren

```
// main.js  
require(['antwort', 'jq'], function(antwort, $){  
    $('#Foo').html(antwort);  
});
```

```
// In der App modul42.js und jQuery laden
require(['modul42', 'jquery'], function(modul42){

  // In modul42 liegt die Modul-API
  var value = modul42.doSomething();
  alert(value);

});
```



```
// Moduldefinition von modul42.js
define(function(){

  // Return des Callbacks = Modul-API
  return {
    doSomething: function(){
      return 42;
    }
  };

});
```

Fazit AMD

- JavaScript hat noch kein in heutigen Browsern benutzbares Modulsystem
- DIY-Lösung im Frontend: AMD + RequireJS

Alles klar zu AMD?

2. QUnit

Frontend Unit Testing

QUnit

- **JS-Testframework**, u.A. verwendet von jQuery
- Einfach, browserbasiert
- Alternativen: Jasmine, Mocha, viele weitere



// Unser kleines Testprojekt

```
function ltrim(str){  
  if(typeof str !== 'string'){  
    throw new TypeError();  
  }  
  return str.replace(/^s+/, '');  
}
```

```
ltrim('  Hallo !  '); // > 'Hallo !  '
```

```
<!DOCTYPE html>
<meta charset="utf-8">
<title>Testseite</title>

<link rel="stylesheet" href="qunit.css">
<script src="qunit.js"></script>

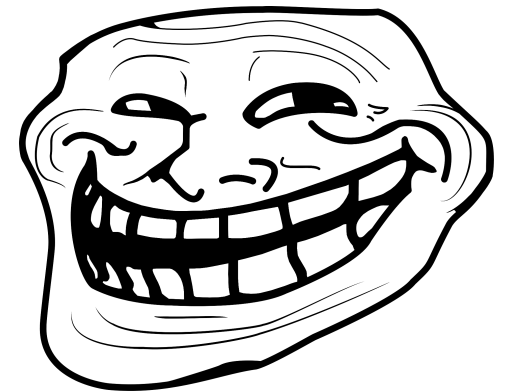
<div id="qunit"></div>
<div id="qunit-fixture"></div>

<script src="ltrim.js"></script>
<script src="test.js"></script>
```

```
QUnit.test('ltrim() kürzt Strings', function(assert){  
    var str = '  Hallo !  ';  
    var expected = 'Hallo !  ';  
    var actual = ltrim(str);  
    assert.equal(actual, expected);  
});
```

```
QUnit.test('ltrim() nimmt nur Strings', function(assert){  
    var objects = [ {}, [], document.createElement('div') ];  
    objects.forEach(function(object){  
        assert.throws(function(){  
            ltrim(object);  
        }, Object.prototype.toString.call(object));  
    });  
});
```

```
var str = new String('x');  
typeof str; // > "object"  
ltrim(str) // TypeError!
```



```
QUnit.test('ltrim() nimmt String-Objekte', function(assert){  
    var str = new String('  Hallo !  ');  
    var expected = 'Hallo !  '  
    var actual = ltrim(str);  
    assert.equal(actual, expected);  
});
```

```
function ltrim(str){  
  if(typeof str !== 'string' && str != str.toString()){  
    throw new TypeError();  
  }  
  return str.replace(/^s+/, '');  
}
```

Assertions in QUnit

- `strictEqual(a, b)`, `equal(a, b)`, `notEqual(a, b)`
- `deepEqual(a, b)`, `notDeepEqual(a, b)`
- `propEqual(a, b)`
- `throws(fn)`
- `ok(x)`

// Asynchrone Funktion

```
function foo(callback){  
  setTimeout(function() {  
    callback(42);  
  }, 1000);  
}
```

// Asynchroner Test (neu)

```
QUnit.test('foo() ergibt 42', function(assert){  
  var done = assert.async();  
  foo(function(result){  
    assert.equal(result, 42);  
    done(); // Ende des Tests signalisieren  
  });  
});
```

How To gut testbarer Code

- Modular bleiben
- Klare Trennung von Zuständigkeiten
- Funktionen ohne Nebenwirkungen

Motto: **KISS!**

3. Promises

Asynchrone Programmierung mit q.js

Asynchrones JavaScript

JavaScript ist single-threaded. Blockierende Prozesse wie IO oder Geolocation werden über Callback-Funktionen abgewickelt.

// Schön wäre:

```
var position = geolocation.getCurrentPosition();
```

// Realität ist:

```
geolocation.getCurrentPosition(function(position){  
});
```

Was sind Promises?

- Promise = *“an object representing a value given by a deferred computation”*
- Universelle Kapselung asynchroner Operationen
- Bestes Beispiel: **jQuery***

* Eigentlich das schlechteste Beispiel; Details später

```
// Herkömmlicher Callback  
$ajaxPromise.then(function(data){  
    console.log(data);  
});Callback hinzufügen  
ajaxPromise.then(function(result){  
    console.log(result);  
});
```

```
// Weiteren Callback hinzufügen  
ajaxPromise.then(function(){  
    window.alert('Fertig!');  
});
```

Einfache Benutzung

1. Promise wird erstellt (z.B. mit `$.get ()`; Status ist *promise pending*)
2. Async-Operation wird entweder ein Erfolg (*promise fulfilled/resolved*) oder ein Fehlschlag (*promise rejected*)
3. Die mit Promise-Methoden wie `then ()` hinzugefügten Callbacks werden ausgeführt


```
// Promise mit Fehler-Callback  
var ajaxPromise = $.get('/');  
  
ajaxPromise.then(function(data){  
    console.log(data);  
}, function(err){  
    console.error(err);  
});
```

Vorteile

1. Einheitliche API für alle asynchronen Operationen
2. Benutzung von Promises vor Abschluss der Operation
3. Einfache, lesbare Verkettungen
4. Aggregation in Master-Promises
5. Cross-Library-Kompatibilität

```
/* Promises verketten */
```

```
function loadAjaxInto(selector){  
  var $container = $(selector);  
  // Schritt 1: Container ausblenden  
  return $container.hide('slow').promise()  
  // Schritt 2: Request absetzen  
  .then(function()){  
    return $.get('/');  
  })  
  // Schritt 3: Container anzeigen  
  .then(function(response){  
    return $container.html(response).show('slow').promise();  
  });  
}
```

```
// Ladesequenz starten
```

```
loadAjaxInto('#Container').then(...);
```

```
/* Preisfrage: Wie kann man hier am einfachsten einen  
vierten Schritt einbauen? */
```

```
/* Promises aggregieren (bzw.  
Master-Promise erzeugen) */
```

```
$.when(  
    $.get('/foo'),  
    $.get('/bar'),  
    $.get('/baz')  
).then(function(resultArray){  
    window.alert('Alle fertig!');  
});
```

```
/* when() ist ein üblicher Bestandteil vieler  
Promise-Libraries */
```

Besser als jQuery: [q.js](#)

- Standardkonforme Implementierung von Promises/A+
- Etwas andere API, gleiche Funktionalität
- Viele, viele Promise-Funktionen
- Für uns relevant: **Erzeugen und verarbeiten**

```
/* Deferred-Objekt = Promise-Vorstufe */
```

```
var deferred = Q.defer();
```

```
/* > {  
  resolve: [Function],  
  reject: [Function],  
  then: [Function]  
} */
```

```
var promise = deferred.promise;
```

```
/* > {  
  resolve: undefined,  
  reject: undefined,  
  then: [Function]  
} */
```

// Eine Promises produzierende Funktion

```
function createPromise(){  
  var deferred = Q.defer();  
  var success = (Math.random() < 0.5);  
  setTimeout(function() {  
    if(success){  
      deferred.resolve('Epic win!');  
    }  
    else {  
      deferred.reject('Epic fail!');  
    }  
  }, 1000);  
  return deferred.promise;  
}
```

Kleine Übung

AMD + jQuery + q.js + QUnit

Wir bauen ein testbares AMD-Modul!

1. Datei `modul.js` enthält ein AMD-Modul
2. Das Modul stellt eine Funktion bereit
3. Die Funktion gibt ein Promise zurück
4. Promise wird mit den Wert `42` aufgelöst, wenn ein Button angeklickt wird

Verwendete Hilfsmittel: jQuery, q.js

Dateiübersicht unterlagen/javascript/

- require.js
- qunit.css, qunit.js
- jquery.js, q.js
- [test.html](#), **test.js** (nur anschauen)
- **modul.js** (bearbeiten)
- Spickzettel-PDF

Wir bauen ein testbares AMD-Modul!

- Vorlage: `unterlagen/javascript`
- Aufgabe: `modul.js` so schreiben, dass es den Test in `test.html` besteht
- Spickzettel: `spickzettel.pdf`

Tests bearbeiten verboten, Nachfragen erlaubt!