JQUERY

Einführung

Viele Aufgaben, die man mittlerweile relativ einfach mit JavaScript durchführen kann, waren lange Zeit aufgrund von Browserunterschieden nur mit verhältnismäßig vielen Quelltext möglich. Aus diesem Grund haben sich verschiedene Bibliotheken herausgebildet, die verschiedene Aufgaben wir das Arbeiten mit dem DOM vereinfachen. Eine der bekanntesten dieser Bibliotheken ist jQuery, welches auch heutzutage in den Werkzeugkoffer eines jeden Webentwicklers gehört

Eine der wohl bekanntesten JavaScript-Bibliotheken ist die Bibliothek jQuery, die das Arbeiten mit JavaScript teilweise stark vereinfacht. Auch wenn mittlerweile viele Dinge auch mit Standardmethoden der DOM API möglich sind, ist jQuery nach wie vor eine ernst zu nehmende Bibliothek.

Die Bibliothek jQuery ist so umfangreich, das hier nur auf eine Auswahl von Aspekten eingegangen wird.

Die Bibliothek jQuery abstrahiert browserspezifische Details und bietet eine einheitliche Schnittstelle. Im Wesentlichen hat jQuery also folgende Vorteile:

- **Einfacheres Arbeiten mit dem DOM:** jQuery vereinfacht den Zugriff auf Elemente des DOM-Baumes, indem es verschiedene Helfermethoden bereitstellt. Standardmethoden der aktuellen DOM API wie `querySelector()` und `querySelectorAll()` (die in älteren Browsern nicht zur Verfügung stehen) basieren auf Ideen von jQuery
- **Einfacheres Arbeiten mit Events:** jQuery vereinfacht das Arbeiten mit Events und bietet hierzu entsprechende Helfermethoden an.
- **Einfacheres Formulieren von Ajax-Anfragen:** jQery vereinfacht das Formulieren von Ajax-Anfragen, indem auch hier browserspezifische Details verborgen werden.

CODERS.BAY

jQuery ist nicht immer notwendig

Auch wenn jQuery eine wirklich mächtige Bibliothek ist, solltest du nicht den Fehler begehen, jQuery mit JavaScript gleichzusetzen und zuerst jQuery und dann die Sprache JavaScript zu lernen. jQuery kann sicherlich in vielen Fällen eine Unterstützung sein, oftmals ist die Verwendung der Bibliothek aber auch gar nicht notwendig, weil man entsprechende Aufgaben bereits mit reinem JavaScript-Code oder aber anderen, schlankeren Bibliotheken lösen kann.
Websites wie *You might not need jQuery (http://youmightnotneedjquery.com)* demonstrieren dies anhand verschiedener Beispiele

Tipp

Für einen JavaScript-Entwickler ist es prinzipiell nicht schlecht, beides zu können: sowohl Bibliotheken wie jQuery einsetzen können als auch die grundlegenden Sprachkonzepte sicher beherrschen.

jQuery einbinden

CODERS.BAY

Die Bibliothek jQuery kann auf verschiedene Weisen eingebunden werden. Unter *https://jquery.com/download/* kannst du die jeweils aktuelle Version der Bibliothek herunterladen, wobei sowohl die "normale" Version zum Download bereitsteht als auch eine minifizierte (sprich komprimierte) Version, welche bezüglich der Dateigröße möglichst klein ist. Nachdem du die jeweilige Datei heruntergeladen hast, kannst du diese wie gewohnt über das `<script>`-Element einbinden

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>jQuery Beispiel</title>
        <link rel="stylesheet" href="styles/main.css">
    </head>
    <body>
       <script src="scripts/jquery-3.0.0.min.js"></script>
       <script src="scripts/main.js"></script>
    </body>
</html>
```

Minifizierte Versionen vs. nicht-minifizierte Versionen

Die meisten Bibliotheken bieten sowohl eine "normale" (sprich nicht minifizierte Version) zum Download an als auch eine minifizierte Version. In Letzterer sind beispielsweise Leerzeichen entfernt, oft auch Kommentare innerhalb des Codes, und vieles mehr ist dahingegen optimiert, dass die Dateigröße verringert und somit die Downloadzeit reduziert wird. Folglich sind minifizierte Versionen von Bibliotheken für den Einsatz in einem Produktivsystem geeignet. Die nicht minifizierte Version eignet sich eigentlich nur dann, wenn du während der Entwicklung auch einen Blick in die entsprechende Bibliothek werfen möchtest. beispielsweise während des Debuggens.

jQuery über ein CDN einbinden

CODERS.BAY

Alternativ gibt es die Möglichkeit, jQuery über ein sogenanntes Content Delivery Network (kurz CDM) einzubinden. Den entsprechenden Link zur aktuellen Version gibt es unter https://code.jquery.com/

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>jQuery Beispiel</title>
        <link rel="stylesheet" href="styles/main.css">
    </head>
    <body>
        <script
        src="https://code.jquery.com/jquery-3.4.1.min.js"
        integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo="
        crossorigin="anonymous"></script>
       <script src="scripts/main.js"></script>
    </body>
</html>
```

Content Delivery Network (CDN)

Bei einem *Content Delivery Network* (auch *Content Distribution Network*, kurz: *CND*) handelt es sich um ein Netzwerk von Servern, die über das Internet verbunden sind und die Anfragen so verteilen, dass sie möglichst schnell beantwortet werden können. In der Regel spielt dabei der geografische Ort einer Anfrage eine große Rolle: Greift beispielsweise ein Nutzer aus Deutschland auf die Webseite zu und dort ist jQuery über eine CDN-URL eingebunden, wird der entsprechende Code von einem Server aus Deutschland an den Nutzer gesendet. Ein Nutzer, der aus den USA auf die Webseite zugreift, wird dagegen von einem dort ansässigen Server beliefert.

jQuery verwenden

Das Kernstück von jQuery bildet die Funktion **`jQuery()`** bzw. die gleichbedeutende Shortcut-Funktion **`$()`** (im Folgenden *jQuery-Methode* genannt). Diese Funktion kann mit verschiedenen Argumenten aufgerufen werden, von denen drei Formen besonders häufig verwendet werden:

Aufruf mit CSS-Selektor: Hierbei wird der jQuery-Methode ein Selektor (ähnlich den bereits bekannten CSS-Selektoren) übergeben und erhält als Rückgabewert ein Objekt, welches die Elemente der Webseite enthält, die auf den Selektor zutreffen.

Beispiel: Das von der Methode zurückgegebene Objekt stellt ein sogenanntes Wrapperobjekt (im Folgenden *jQuery-Objekt* genannt) für die entsprechenden Elemente dar und stellt für diese Elemente verschiedene Methoden zur Verfügung

```
let selectedElements = jQuery('body > div > span');

let selectedElements = $('body > div > span');
```

Aufruf mit Knoten aus dem DOM-Baum: Alternativ zu dem Aufruf mit einem Selektor kann die jQuery-Methode auch mit einem Knoten des DOM-Baumes aufgerufen werden bzw. mit dem entsprechenden den jeweiligen Knoten repräsentierenden JavaScript-Objekt. Auch in diesem Fall stellt das jQuery-Objekt ein Wrapperobjekt um den übergebenen Knoten dar und bietet zusätzliche Methoden an. Um beispielsweise einen Event-Listener zu definieren, der aufgerufen wird, wenn das **`document`**-Objekt vollständig geladen wurde, siehst du im Beispiel. Die Methode **`ready()`** gibt es für das **`document`**-Objekt ja bekanntermaßen nicht, sondern wird erst durch das jQuery-Objekt indirekt zur Verfügung gestellt

```
$(document).ready(() => {
    console.log('Webseite geladen')
;
});
```

Aufruf mit HTML-String: Es können die jQuery-Methoden auch dazu verwendet werden, um neue Elemente zu erzeugen. Dazu wird der Methode einfach der entsprechende HTML-Code für das zu erstellende Element als Zeichenkette übergeben.

```
let newElement = $('<div>Neues Element</div
>');
```

Aufgaben mit jQuery vereinfachen

CODERS.BAY

Dass jQuery das Arbeiten mit dem DOM vereinfacht, lässt sich am besten anhand eines Beispiels demonstrieren. Angenommen, du hast eine HTML-Liste, in der jeder Listeneintrag eine URL enthält (als Text und nicht als Link), und due möchtest gerne mithilfe von JavaScript zur Laufzeit aus den URLs echte Links erstellen. Mit anderen Worten: Der Textinhalt der Listeneinträge soll in `<a>`-Elemente umgewandelt werden.

Unter Verwendung von reinem JavaScript dient das erste Beispiel. Zunächst müssten die entsprechenden `<li>`-Elemente ausgewählt werden, dann muss jeweils der Textinhalt extrahiert und entfernt, ein neues `<a>`-Element erstellt, dessen `href`-Attribut und Textinhalt gesetzt sowie das Element dem `<li>`-Element als Kindelement hinzugefügt werden

```
// JavaScript
'use strict';
function init() {
    let listItems = document.getElementsByTagName('li
');
    for(let i = 0; i < listItems.length; i++) {
        let listItem = listItems[i];
        let url = listItem.textContent;
        listItem.textContent = '';
        let link = document.createElement('a');
        link.setAttribute('href', url);
        let linkText = document.createTextNode(url);
        link.appendChild(linkText);
        listItem.appendChild(link);
    }
}
```

```
// JavaScript
'use strict';
function init() {
    let listItems = document.getElementsByTagName('li');
    for(let i = 0; i < listItems.length; i++) {
        listItems[i].innerHTML = '<a href="'
        + listItems[i].textContent + '">'
        +listItems[i].textContent + '</a>';
    }
}
```

Aufgaben mit jQuery vereinfachen

CODERS.BAY

Mit jQuery wird das Ganze etwas eleganter. Hier kommt die Methode **`wrapInner()`** zum Einsatz, die durch das jQuery-Objekt den ausgewählten Elementen quasi zur Verfügung gestellt wird. Diese Methode umgibt den Inhalt der ausgewählten Elemente mit dem HTML-Code, der im Beispiel von der übergebenen Funktion zurückgegeben wird. Viel einfacher als der Code von eben

```
// jQuery
'use strict';
function init() {
    $('li').wrapInner(
        function() {
            return '<a href"' + this.textContent + '"></a>';
        }
    );
}
```

jQuery-Plugins

Solltest du eine Funktionalität innerhalb von jQuery übrigens mal nicht vorfinden, stehen dir im Internet Tausende von Plugins zur Verfügung. Eine sehr gute Übersicht gibt die offizielle jQuery Registry unter *https://plugins.jquery.com*

DOM: Elemente selektieren

CODERS.BAY

Mithilfe von jQuery lassen sich Elemente unter Verwendung CSS-ähnlicher Selektoren auswählen. Diese Selektoren lassen sich in folgende Gruppen einteilen:

- Basisselektoren: Hierbei handelt es sich um grundlegende Selektoren, wie sie aus dem CSS bekannt sind
- Hierarchieselektoren: Selektoren, welche die Hierarchie von Elementen einbeziehen.
- Basisfilterselektoren: Selektoren, mit denen einzelne Elemente gezielter herausgefiltert werden können
- Inhaltsfilterselektoren: Selektoren, die den Inhalt von Elementen mit einbeziehen
- Sichtbarkeitsfilterselektoren: Selektoren bezüglich der Sichtbarkeit von Elementen
- Attributfilterselektoren: Selektoren: welche die Attribute von Elementen mit einbeziehen
- Formularfilterselektoren Selektoren, die gezielt für die Selektion von Formularelementen nützlich sind
- Kindfilterselektoren: Selektoren für das Selektieren von Kindelementen

Die einzelnen Selektoren können selbstverständlich untereinander kombiniert werden.

```
$(document).ready(() => {
    let inputElements = $('input');          // alle <input>-Elemente
    let max = $('#max');                     // Elemente mit ID "max"
    let oddElements = $('.odd');             // Elemente mit Klasse "odd"
    let elements = $('td, th');              // alle <td>- und <th>-Elemente</th></td>
    let inputMax = $('input[name="max"');    // <input>-Elemente, deren name-
                                                Attribut den Wert "max" hat
    let oddRows = $('tr:odd');               // alle "ungeraden" <tr>-Elemente
    let evenRows = $('tr:even');             // alle "geraden" <tr>-Elemente
    let listItemAtIndex = $('li:eq(2)');     // alle <li>-Elemente an Index2
    let allOthers = $(':not(li)');           // alle Nicht-<li>-Elemente
    let notExample = $(':not(.example');     // alle Elemente ohne CSS-Klasse "example"
});
```

Basisselektoren in jQuery

CODERS.BAY

Selektor	Beschreibung
`*`	Elemente mit beliebigem Elementnamen
`elementName`	Elemente vom Typ `elementName`
`id`	Element mit der ID `id`
`.class`	Elemente mit der Klasse `class`
`selektor1, selektor2`	Elemente, die entweder auf den Selektor `selektor1` oder den Selektor `selektor2` passen

Hierarchieselektoren in jQuery

CODERS.BAY

Selektor	Beschreibung
`element1 element2`	Alle Elemente vom Typ `element2`, die sich innerhalb eines Elements vom Typ `element1` befinden
`element1 > element2`	alle Elemente vom Typ `element2`, die direkte Kindelemente eines Elements vom Typ `element1` sind
`element1 + element2`	alle Elemente vom Typ `element2`, die direkt auf ein Element vom Typ `element1` folgen
`element1 ~ element2`	alle Elemente vom Type `element2`, die auf ein Element vom Type `element1` folgen

Basisfilterselektoren in jQuery

CODERS.BAY

Selektor	Beschreibung
`:animated`	Selektiert Elemente, die zurzeit im Rahmen einer Animation verwendet werden
`:eq()`	Selektiert das Element an einem bestimmten Index (welcher als Argument übergeben wird)
`:even`	Selektiert alle geraden Elemente
`:first`	Selektiert das erste Element
`:gt()`	Selektiert alle Elemente, die einen größeren als den als Argument übergebenen Index haben
`:header`	Selektiert alle Überschriftenelemente, also `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>` und `<h6>`
`:lang()`	Selektiert alle Elemente für die übergebene Sprache
`:last`	Selektiert das letzte Element
`:lt()`	Selektiert alle Elemente, die einen kleineren als den als Argument übergebenen Index haben
`:not()`	Selektiert alle Elemente, welche nicht auf den übergebenen Selektor zutreffen
`:odd`	Selektiert alle ungeraden Elemente
`:root`	Selektiert das Wurzelelement (nicht den Dokumentenknoten), sprich das `<html>`-Element
`:target`	Selektiert das Element, welches durch die Fragment-ID der entsprechenden URL identifiziert wird. Lautet die URL bspw.: *http://www.test.at#jquery*, würde mit `$(':target')` das Element mit der ID `jquery` selektiert

Inhaltsfilterselektoren in jQuery

CODERS.BAY

Selektor	Beschreibung
`:constains()`	Selektiert alle Elemente, die den übergebenen Text enthalten
`:empty`	Selektiert alle Elemente, die keine Kindelemente bzw. Kindknoten haben
`:has()`	Selektiert alle Elemente, die mindestens ein Element enthalten, welches auf den übergebenen Selektor zutrifft
`:parent`	Selektiert alle Elemente, die mindestens einen Kindknoten haben

Sichtbarkeitsfilterselektoren in jQuery

CODERS.BAY

Selektor	Beschreibung
`:hidden`	Selektiert alle nicht sichtbaren Elemente
`:visible`	Selektiert alle sichtbaren Elemente

Attributfilterselektoren in jQuery

CODERS.BAY

Selektor	Beschreibung
`[name |= "value"]`	Selektiert Elemente, deren Wert des Attributes `name` eine Reihe von mit Minuszeichen getrennten Werten sind, wobei der erste Wert `value` ist
`[name*= "value"]`	Selektiert Elemente mit Attribut `name`, dessen Wert `value` als Substring enthält
`[name~= "value"]`	Selektiert Elemente mit Attribut `name`, welches als Wert eine Liste von Werten hat, von denen eines den Wert `value` hat
`[name$= "value"]`	Selektiert Elemente mit Attribut `name`, dessen Wert mit `value` endet
`[name= "value"]`	Selektiert Elemente mit Attribut `name`, welches den Wert `value` hat
`[name!= "value"]`	Selektiert Elemente mit Attribut `name`, welches nicht den Wert `value` hat
`[name^= "value"]`	Selektiert Elemente mit Attribut `name`, dessen Wert mit `value` beginnt
`[name]`	Selektiert Elemente mit Attribut `name`
`[name= "value]` `[name2= "value2]`	Selektiert Elemente mit Attribut `name`, welches den Wert `value` hat und mit Attribut `name2`, welches den Wert `value2` hat

Formularfilterselektoren in jQuery

CODERS.BAY

Selektor	Beschreibung
`:button`	Selektiert alle Schaltflächen
`:checkbox`	Selektiert alle Checkboxen
`:checked`	Selektiert alle ausgewählte bzw. aktivierte Elemente
`:disabled`	Selektiert alle deaktivierte Elemente
`:enabled`	Selektiert alle aktivierten Elemente
`:focus`	Selektiert alle Elemente, die den Fokus haben
`:file`	Selektiert alle Dateieingabefelder
`:image`	Selektiert alle Elemente, deren `type`-Attribute den Wert `image` hat
`:input`	Selektiert alle Eingabefelder
`:password`	Selektiert alle Passwortfelder
`:radio`	Selektiert alle Radiobuttons
`:reset`	Selektiert alle Elemente, deren `type`-Attribut den Wert `reset` hat

Formularfilterselektoren in jQuery

CODERS.BAY

Selektor	Beschreibung
`:selected`	Selektiert alle ausgewählten Elemente
`:submit`	Selektiert alle Elemente, deren type-Attribut den Wert submit hat
`:text`	Selektiert alle Texteingabefelder

Kindfilterselektoren in jQuery

CODERS.BAY

Selektor	Beschreibung
`:first-child`	Selektiert das erste Kindelement
`:first-of-type`	Selektiert das erste Element von einem bestimmten Typ
`:last-child`	Selektiert das letzte Kindelement
`:last-of-type`	Selektiert das letzte Element von einem bestimmten Typ
`:nth-child()`	Selektiert das n-te Kindelement
`:nth-last-child()`	Selektiert das n-letzte Kindelement
`:nth-of-type()`	Selektiert das n-te Element von einem bestimmten Typ
`:nth-last-of-type()`	Selektiert Elemente, die das einzige Kindelement ihres Elternelements sind
`:only-of-type()`	Selektiert Elemente, die das einzige Kindelement ihres Elternelements von einem bestimmten Typ sind

Auf Inhalte zugreifen und diese verändern

CODERS.BAY

Nach dem auswählen, stehen verschiedene Methoden zur Verfügung, mit denen man auf den Inhalt zugreifen und diesen verändern kann

- Zugriff auf und Ändern von HTML- und Textinhalte
- Hinzufügen von Inhalten innerhalb eines Elements
- Hinzufügen von Inhalten außerhalb eines Elements
- Hinzufügen von Inhalten um ein Element herum
- Ersetzen von Inhalten
- Entfernen von Inhalten

Über die Methode `html()` kann beispielsweise auf den HTML-Inhalt zugegriffen werden, über die Methode `text()` auf den Textinhalt (jeweils schreibend und lesend), über `append()` wird neuer Inhalt an den bestehenden Inhalt der ausgewählten Elemente hinzugefügt, über `prepend()` entsprechend vor den bestehenden Inhalt, über `after()` wird neuer Inhalt hinter die ausgewählten Elemente hinzugefügt und mit `before()` vor die ausgewählten Elemente.

Beispiele für das Ändern und Hinzufügen von Inhalten

CODERS.BAY

```
// Hinzufügen von neuem HTML-Inhalt
$('#main').html('<div>Neuer Inhalt</div>');
// Zugriff auf den HTML-Inhalt
let htmlContent = $('#main').html();

// Hinzufügen von neuem Textinhalt
$('#main').text('<div>Neuer Inhalt</div>');
// Zugriff auf den Textinhalt
let htmlContent = $('#main').text();

/**
 * Hinzufügen von neuem Inhalt hinter den
 * bestehenden Inhalt jedes <div>-Elements
 * mit der CSS-Klasse "example"
 */
$('div.example').append('<p>Beispiel</p>');

/**
 * Hinzufügen von neuem Inhalt vor den
 * bestehenden Inhalt jedes <div>-Elements
 * mit der CSS-Klasse "example"
 */
$('div.example').prepend('<p>Beispiel</p>');

/**
 * Hinzufügen von neuem Inhalt hinter jedes
 * <div>-Elements mit der CSS-Klasse "example"
 */
$('div.example').after('<p>Beispiel</p>');

/**
 * Hinzufügen von neuem Inhalt vor jedes
 * <div>-Elements mit der CSS-Klasse "example"
 */
$('div.example').before('<p>Beispiel</p>');
```

Methoden für das Abrufen und Definieren von Inhalten

CODERS.BAY

Selektor	Beschreibung
`html()`	Ohne Argument liefert diese Methode den HTML-Inhalt eines Elements Mit Argument setzt diese Methode den HTML-Inhalt eines Elements
`text()`	Ohne Argument liefert diese Methode den Textinhalt eines Elements. Mit Argument setzt diese Methode den Textinhalt eines Elements

Methoden für das Hinzufügen von Inhalten innerhalb eines Elements

CODERS.BAY

Selektor	Beschreibung
`append()`	Fügt Inhalt an das Ende der ausgewählten Elemente ein: `$(a).append(b)` fügt dem Element `a` den Inhalt `b` an das Ende hinzu
`appendTo()`	Umgekehrt wird `append()`, d. h., `$(a).appendTo(b)` fügt das Element `a` als Inhalt dem Element `b` an das Ende hinzu
`prepend()`	Fügt Inhalt an den Anfang der ausgewählten Elemente ein: `$(a).prepend(b)` fügt dem Element `a` den Inhalt `b` an den Anfang hinzu
`prependTo()`	Umgekehrt wird `prepend()`, d. h., `$(a).prependTo(b)` fügt das Element `a` als Inhalt dem Element `b` an den Anfang hinzu

Methoden für das Hinzufügen von Inhalten außerhalb eines Elements

CODERS.BAY

Selektor	Beschreibung
`after()`	Fügt Inhalt hinter jedes der ausgewählten Elemente hinzu: `$(a).after(b)` fügt den Inhalt `b` hinter das Element `a` ein
`before()`	Fügt Inhalt vor jedes der ausgewählten Elemente hinzu: `$(a).before(b)` fügt den Inhalt `b` vor das Element `a` ein
`insertAfter()`	Umgekehrt wie `after()`, d. h., `$(a).insertAfter(b)` fügt das Element `a` nach dem Element `b` hinzu
`insertBefore()`	Umgekehrt wie `before()`, d. h., `$(a).insertBefore(b)` fügt das Element `a` vor dem Element `b` hinzu

Methoden für das Hinzufügen von Inhalten um ein Element

CODERS.BAY

Selektor	Beschreibung
`clone()`	Erstellt eine Kopie der ausgewählten Elemente. Genauer gesagt, wird dabei eine sogenannte *tief gehende Kopie (Deep Copy)* erstellt, die auch Kindelemente der ausgewählten Elemente kopiert
`wrap()`	Fügt um jedes der ausgewählten Elemente herum jeweils neue Inhalte hinzu
`wrapAll()`	Fügt um alle ausgewählten Elemente herum neuen Inhalt hinzu
`wrapInner()`	Fügt um den Inhalt jedes der ausgewählten Elemente herum jeweils neuen Inhalt hinzu

Methoden für das Ersetzen von Inhalten

CODERS.BAY

Selektor	Beschreibung
`replaceAll()`	Umgekehrt wie `replaceWith()`, d. h., `$(a).replaceAll(b)` ersetzt alle Elemente, die durch den Selektor `b` selektiert werden, durch den Inhalt in `a`
`replaceWith()`	Ersetzt die ausgewählten Elemente durch einen neuen Inhalt: `$(a).replaceWith(b)` ersetzt das Element `a` durch den Inhalt `b`

Methoden für das Entfernen von Inhalten

CODERS.BAY

Selektor	Beschreibung
`detach()`	Entfernt die ausgewählten Elemente aus dem DOM-Baum, behält dabei aber Referenzen auf die entfernten Elemente, sodass diese zu einem späteren Zeitpunkt wieder in den DOM-Baum eingebaut werden können
`emtpy()`	Entfernt von den ausgewählten Elementen jeweils alle Kindknoten
`remove()`	Entfertn die ausgewählten Elemente aus dem DOM-Baum
`unwrap()`	Entfernt von den ausgewählten Elementen jeweils das Elternelement

Ausgewählte Elemente filtern

CODERS.BAY

Das jQuery-Objekt stellt für die ausgewählten Elemente ebenfalls verschiedene Methoden bereit, mithilfe derer die aktuell ausgewählten Elemente weiter eingegrenzt werden können.

```
// Auswahl des dritten <li>-Elements
$('li').eq(2);
// Auswahl des ersten <li>-Elements
$('li').first();
// Auswahl der <li>-Elemente, welche die CSS-Klasse ".selected" haben
$('li').filter('selected');
// Auswahl aller <li>-Elemente, die ein <ul>-Element enthalten
$('li').has('ul');
// Auswahl aller Elemente, welche die CSS-Klasse ".selected" haben
$('li').has('.selected');
// Auswahl des letzten <li>-Elements
$('li').last();
// Auswahl aller css-Attribute der <li>-Elemente
$('li').map(() => { this.className });
// Auswahl aller <li>-Elemente, welche nicht die CSS-Klasse ".selected" haben
$('li').not('.selected');
// Auswahl der ersten zwei <li>-Elemente
$('li').slice(0, 2);
```

Methoden für das Filtern

CODERS.BAY

Selektor	Beschreibung
`add()`	Fügt einer Auswahl von Elementen neue Elemente hinzu
`addBack()`	Fügt einer vorigen Auswahl der aktuellen Auswahl von Elementen hinzu
`eq()`	Reduziert die ausgewählten Elemente auf ein Element an einem bestimmten Index
`filter()`	Reduziert die ausgewählten Elemente auf die Elemente, die (1) auf den übergebenen Selektor zutreffen oder (2) für die die übergebene Filterfunktion `true` zurückgibt
`find()`	Selektiert die Nachfahrenelemente der ausgewählten Elemente, für die (1) der übergebene Selektor zutrifft, (2) welche in dem übergebenen jQuery-Objekt enthalten sind oder(3) welche gleich dem übergebenen Element sind
`first()`	Reduziert die ausgewählten Elemente auf das erste Element
`has()`	Reduziert die ausgewählten Elemente auf die Elemente, die ein Nachfahrenelement haben, welches (1) auf den übergebenen Selektor zutrifft oder (2)gleich dem übergebenen Element ist.
`is()`	Prüft, ob mindestens eines der ausgewählten Elemente (1) auf den übergebenen Selektor zutrifft, (2) für eines von ihnen die übergebene Filterfunktion den Wert `true` zurückgibt, (3) eines der Elemente in dem übergebenen jQuery-Objekt enthalten ist oder (4) es sich bei einem der ausgewählten Elemente um eines der als Parameter übergebenen Elemente handelt
`last()`	Reduziert die ausgewählten Elemente auf das letzte Element
`not()`	Reduziert die ausgewählten Elemente auf die Elemente, die (1) nicht auf den übergebenen Selektor zutreffen, (2) nicht die übergebene Filterfunktion erfüllen oder (3) nicht in dem übergebenen jQuery-Objekt enthalten sind
`slice()`	Reduziert die ausgewählten Elemente auf eine durch Start- und Endindex definierte Teilmenge

Auf Attribute zugreifen

CODERS.BAY

Das jQuery-Objekt bietet einige Methoden an, um auf HTML-Attribute zugreifen zu können.

```
let element = $('a#main');

// lesender Zugriff auf das Attribut "href" des Elements
let href = element.attr('href');

// schreibender Zugriff auf das Attribut "href" des Elements
element.attr('href', 'index.html');

// alternativer schreibender Zugriff über Konfigurationsobje
kt
element.attr({
    href: 'index.html',
    target: '_blank'
});

//Entfernen des Attributes "href" aus dem Element
element.removeAttr('href');

// Hinzufügen einer CSS-Klasse
element.addClass('highlighted');

// Entfernen einer CSS-Klasse
element.removeClass('hilighted');
```

Methoden für den Zugriff auf Attribute und CSS-Klassen

CODERS.BAY

Selektor	Beschreibung
`attr()`	Mit einem Argument liefert diese Methode den Wert zu einem Attribut (z. B. `$('a#main.attr('href')`). Mit zwei Argumenten setzt diese Methoden den Wert eines Attributes (z. B. `$('a#main').attr('href', 'index.html');`
`removeAttr()`	Entfernt ein Attribut von einem Element, z. B. `$('a#main').removeAttr('href')`).
`addClass()`	Fügt eine neue CSS-Klasse den Werten in dem `class`-Attribut hinzu. In neueren Browsern ist dies dank der standardisierten Eigenschaft `classList` und ihrer Methode `add()` auch ohne jQuery möglich
`removeClass()`	Entfernt eine CSS-Klasse aus den Werten in dem `class`-Attribut. Auch dies ist in neueren Browsern dank der Eigenschaft `classList` und ihrer Methode `remove()` ohne jQuery möglich. Allerdings kann der Methode `removeClass()` alternativ auch eine Funktion übergeben werden, die eine kommaseparierte Liste von CSS-Klassen zurückgibt. Das ist über die Methode `remove()` der `classList`-Eigenschaft wiederum nicht möglich
`toggleClass()`	Schaltet eine CSS-Klasse um: Hat das Element die übergebene Klasse, wird sie entfernt, hat das Element nicht die übergebene Klasse, wird diese hinzugefügt

Auf CSS-Eigenschaften zugreifen

CODERS.BAY

```
// Auslesen der Hintergrundfarbe des <body>-Elements
let backgroundColor = $('body').css('background-color');

// Auslesn der Vordergrundfarbe und der Hintergrundfarbe des <body>-Elements
let properties = $('body').css(['color', 'background-color']);

// Setzen der Hintergrundfarbe des <body>-Elements
$('body').css('background-color', 'blue');

// Setzen der Vordergrundfarbe und der Hintergrundsfarbe des <body>-Elements
$('body').css({
    'color' : 'white',
    'background-color' : 'blue'
});
```

Zwischen Elementen navigieren

CODERS.BAY

```
// Kindelemente
// Auswahl aller Kindelemente von <ul>
let listItems = $('ul').children();
// Auswahl des nächsten Links innerhalb <ul>
let closestLink = $('ul').closest('a');

// Geschwisterlemente
// Auswahl des nächsten Geschwisterlements
let nextSibling = $('ul').next();
// Auswahl des nächsten Linkelements
let nextSiblingLink = $('ul').next('a');
// Auswahl aller nächsten Geschisterelemente
let nextSiblings = $('ul').nextAll();
// Auswahl aller nächsten Linkelemente
let nestSiblingLinks = $('div').nextAll('a');
// Auswahl aller nächsten Geschwisterelemente bis
// zu bestimmtem Element
let nextSiblingsUntil = $('div').nextUntil('a');
// Auswahl des vorigen Geschwisterelements
let previousSibling = $('ul').prev();
// Auswahl aller vorigen Geschwisterlemente
let previousSiblings = $('ul').prevAll();
// Auswahl aller vorigen Geschsterlemenete bis zu bestimmtem El
ement
let previsousSiblingsUntil = $('div').prevUntil('a');
// Auswahl aller Geschisterelemente
let siblings = $('div').siblings();
```

```
// Elternelemente
// Auswahl des Elternelements
let parent = $('ul').parent();
// Auswahl aller Vorfahrenelemente
let parents = $('ul').parents();
// Auswahl aller Vorfahrenelemente bis zu bestimmtem Element
let parentsUntil = $('ul').parentsUntil('div');
```

Methoden für das Navigieren im DOM-Baum

CODERS.BAY

Selektor	Beschreibung
`children()`	Selektiert die Kindelemente der ausgewählten Elemente. Optional kann hierbei ein Selektor übergeben werden, auf welchen die Kindelemente zutreffen müssen.
`closest()`	Selektiert das erste Element der ausgewählten Elemente, das auf den als Parameter übergebenen Selektor zutrifft oder für den eines der Vorfahrenelemente auf den Selektor zutrifft
`next()`	Selektiert das nächste Geschwisterelement der ausgewählten Elemente. Wird dabei ein Selektor übergeben, wird jeweils das nächste Geschwisterelement selektiert, welches auf diesen Selektor zutrifft
`nextAll()`	Selektiert alle nächsten Geschwisterelemente der ausgewählten Elemente. Wird dabei ein Selektor übergeben, werden jeweils die nächsten Geschwisterelemente selektiert, welche auf diesen Selektor zutreffen
`nextUntil()`	Selektiert alle nächsten Geschwisterelemente der ausgewählten Elemente. Wird dabei ein Selektor übergeben, werden jeweils die nächsten Geschwisterelemente bis zu dem Geschwisterelement selektiert, welches auf diesen Selektor zutrifft
`parent()`	Selektiert das Elternelemen der ausgewählten Elemente
`parents()`	Selektiert alle in der Hierarchie vorausgehenden Elternelemente der ausgewählten Elemente

Methoden für das Navigieren im DOM-Baum

CODERS.BAY

Selektor	Beschreibung
`parentsUntil()`	Selektiert alle in der Hierarchie vorausgehenden Elternelemente der ausgewählten Elemente bis zu einem Element, welches (1) auf den übergebenen Selektor zutrifft, (2) dem übergebenen Element entspricht oder (3) in dem übergebenen jQuery-Objekt enthalten ist
`prev()`	Selektiert das vorige Geschwisterelement der ausgewählten Elemente. Wird dabei ein Selektor übergeben, wird jeweils das vorige Geschwisterelement selektiert, welches auf diesen Selektor zutrifft.
`prevAll()`	Selektiert alle vorigen Geschwisterelemente der ausgewählten Elemente. Wird dabei ein Selektor übergeben, werden jeweils die vorigen Geschwisterelemente selektiert, welche auf diesen Selektor zutreffen
`prevUntil()`	Selektiert alle vorigen Geschwisterelemente der ausgewählten Elemente. Wird dabei ein Selektor übergeben, werden jeweils die vorigen Geschwisterelemente bis zu dem Geschwisterelement selektiert, welches auf diesen Selektor zutrifft
`siblings()`	Selektiert alle Geschwisterelemente der ausgewählten Elemente. Wird dabei ein Selektor übergeben, werden jeweils die Geschwisterelemente selektiert, welche auf diesen Selektor zutreffen.

Effekte und Animationen verwenden

CODERS.BAY

```
'use strict';
$(document).reade(() => {
    $('main').animate(
        { opacity: 0.75}, // Eigenschaften
        'fast',           // Geschwindigkeit
        'swing',          // Easing
    () => {
        // Animation abgeschlossen
    });
});
```

Methoden für das Anzeigen und Verstecken von Elementen

CODERS.BAY

Selektor	Beschreibung
`animate()`	Ermöglicht die Animation von CSS-Eigenschaften
`clearQueue()`	Entfernt alle Animationen aus der Warteschlange, die noch nicht ausgeführt wurden
`delay()`	Verzögert eine Animation um eine bestimmte Anzahl Millisekunden
`dequeue()`	Führt die in der Warteschlange nächste Animation aus
`fadeIn()`	Blendet die ausgewählten Elemente ein
`fadeOut()`	Blendet die ausgewählten Elemente aus
`fadeTo()`	Passt die Deckkraft der ausgewählten Elemente an
`fadeToggle()`	Blendet die ausgewählten Elemente je nach Zustand ein oder aus: Ist ein Element sichtbar, wird es ausgeblendet, ist es dagegen nicht sichtbar wird es eingeblendet
`finish()`	Beendete die aktuelle Animation, entfernt alle Animationen aus der Warteschlange und setzt die CSS-Eigenschaften der ausgewählten Elemente auf den Zielwert
`hide()`	Verbirgt die ausgewählten Elemente
`queue()`	Zugriff auf die Animationen in der Warteschlange

Methoden für das Anzeigen und Verstecken von Elementen

CODERS.BAY

Selektor	Beschreibung
`show()`	Zeigt die ausgewählten Elemente
`slideDown()`	Fährt die ausgewählten Elemente von oben nach unten ein
`slideToggle()`	Fährt die ausgewählten Elemente je nach Zustand ein oder aus: Ist ein Element sichtbar, wird es von unten nach oben ausgefahren, ist es dagegen nicht sichtbar, wird es von oben nach unten eingefahren
`slideUp()`	Fährt die ausgewählten Elemente von unten nach oben aus
`stop()`	Stoppt die aktuelle Animation
`toggle()`	Verbirgt oder zeigt die ausgewählten Elemente: Ist ein Element Sichtbar, wird es verborgen, ist es dagegen nicht sichtbar, wird es angezeigt.

Enthält das jQuery-Objekt Verweise auf mehrere Elemente, wird durch Aufruf einer Event-Methode für jedes Element in der Auswahl der entsprechende Event-Listener aufgerufen

```
$('#button').click((event) => {
    console.log('Schaltfläche gedrückt')
;
});
```

Insgesamt lassen sich die Event-Methoden wie folgt einteilen:

- allgemeine Event-Methoden
- Event-Methoden für das Behandeln von allgemeinen Events
- Event-Methoden für das Behandeln von Maus-Events
- Event-Methoden für das Behandeln von Tastatur-Events
- Event-Methoden für das Behandeln von Formular-Events

Methoden für das Verwalten von Event-Handlern

CODERS.BAY

Selektor	Beschreibung
`bind()`	Fügt einen Event-Listener für ein Event hinzu. Seit jQuery 1.7 sollte laut offizieller Dokumentation allerdings die Methode `on()` verwendet werden
`delegate()`	Für ältere jQuery-Versionen ist dies die bevorzugte Methode, um einen Event-Listener für ein Event hinzuzufügen. Seit jQuery 1.7 sollte allerdings die Methode `on()` verwendet werden
`off()`	Entfernt einen Event-Listener für ein Event
`on()`	Fügt einen Event-Listener für ein Event hinzu
`one()`	Fügt einen Event-Listener hinzu, der pro ausgewähltem Element maximal einmal pro Ereignis ausgelöst wird
`trigger()`	Führt alle für ein Event registrierten Event-Listener aus
`triggerHandler()`	Wie die Methode `trigger()`, führt aber nicht das Standardverhalten für ein Even aus (wie beispielsweise das Absenden eines Formulars)
`unbind()`	Entfernt einen Event-Listener für ein Event. Seit jQuery 1.7 sollte laut offizieller Dokumentation allerdings die Methode `off()` verwendet werden
`undelegate()`	Für ältere jQuery-Versionen ist dies die bevorzugte Methode, um einen Event-Listener für ein Event zu entfernen. Seit jQuery 1.7 sollte laut offizieller Dokumentation allerdings die Methode `off()` verwendet werden

Auf allgemeine Ereignisse reagieren

```
$(document).ready(() => {
    console.log('Webseite geladen');
});
```

Verschiedene Methoden für das Registrieren von Event-Listenern

Selektor	Beschreibung
`error()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn ein `error`-Event auftritt
`ready()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn das DOM bzw. das der Methode übergebene Element vollständig geladen wurde
`resize()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn ein `resize`-Event auftritt
`scroll()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn ein Element gescrollt wird

Auf Mausereignisse reagieren

CODERS.BAY

```
$('button#target').click((event) => {
    console.log('Schaltfläche wurde gedrückt
');
});

$('button#target2').click((event) => {
    $('button#target').click();
});
```

Die meisten Methoden in jQuery können mit einer unterschiedlichen Anzahl von Argumenten aufgerufen werden und haben jeweils unterschiedliche Funktionalitäten: Beispielsweise kann mit der Methode `attr()` sowohl HTML-Attribute gelesen als auch geschrieben oder mit den Event-Methoden sowohl Event-Listener registriert als auch Events ausgelöst werden.

Methoden für das Behandeln von Mausereignissen

CODERS.BAY

Selektor	Beschreibung
`click()`	Registrieren von Event-Listenern, die bei Mausklick ausgeführt werden
`dbclick()`	Registrieren von Event-Listenern, die bei doppeltem Mausklick ausgeführt werden
`focusin()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn ein Element den Fokus erhält
`focusout()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn ein Element den Fokus verliert
`hover()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn mit dem Mauszeiger über ein Element "gefahren" wird
`mousedown()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn sich der Mauszeiger über einem Element befindet und die Maustaste gedrückt wurde.
`mouseenter()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn der Mauszeiger ein Element "betritt"
`mouseleave()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn der Mauszeiger ein Element "verlässt"
`mousemove()`	Registrieren von Event-Listenern, die bei Mausbewegungen über einem Element ausgelöst werden
`mouseout()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn der Mauszeiger ein Element "verlässt"
`mouseover()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn der Mauszeiger ein Element "betritt"
`mouseup()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn sich der Mauszeiger über einem Element efindet und die Maustaste losgelassen wurde

Auf Tastaturereignisse reagieren

CODERS.BAY

```
$('input#username')
  .keypress((event) => {
      console.log('Taste zur Eingabe des Nutzernamens gedrückt.');
  })
  .keydown((event) => {
      console.log('Taste wird gedrückt.');
  })
  .keyup((event) => {
      console.log('Taste zur Eingabe des Nutzernamens losgelassen.'
);
  });
```

"Fluent API"

Wenn eine API erlaubt, eine Methode direkt auf dem Rückgabewert einer Methode aufzurufen, spricht man auch von einer *Fluent API*

Methoden für das Behandeln von Tastaturereignissen

CODERS.BAY

Selektor	Beschreibung
`keydown()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn eine Taste auf der Tastatur gedrückt wurde. Wird eine Taste länger gedrückt, wird der Event-Listener mehrmals ausgeführt.
`keypress()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn eine Taste auf der Tastatur gedrückt wurde.
`keyup()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn eine Taste auf der Tastatur losgelassen wurde

Auf Formularereignisse reagieren

CODERS.BAY

```
$('input#username')
  .focus((event) => {
      console.log('Eingabefeld fokussiert.');
  })
  .blur((event) => {
      console.log('Eingabefeld nicht mehr fokussiert.')
;
  })
  .change((event) => {
      console.log('Text geändert');
  })
  .select((event) => {
      console.log('Text ausgewählt');
  });
```

Methoden für das Behandeln von Formularereignissen

CODERS.BAY

Selektor	Beschreibung
`blur()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn ein Formularfeld den Fokus verliert
`change()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn der ausgewählte Wert einer Auswahlliste, einer Checkbox oder einer Gruppe von Radiobuttons geändert wurde
`focus()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn ein Formularfeld den Fokus bekommt
`select()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn der Text eines Eingabefelds (`<input>`-Element vom Typ `text`) oder eines Texteingabebereichs (`<textarea>`-Element) ausgewählt wird
`submit()`	Registrieren von Event-Listenern, die ausgeführt werden, wenn ein Formular abgeschickt wurde

Auf Informationen von Ereignissen zugreifen

CODERS.BAY

```
$('input').on(
    'change',
    {
        value : 4751                        // Datenobjekt
    },
    (event) => {
        console.log(event.currentTarget);   //aktuelles Element
        console.log(event.data);            // Datenobjekt
        console.log(event.data.value);      // Eigenschaft des Datenob
jekts
        console.log(event.pageX);           // x-Position Maus
        console.log(event.pageY);           // y-Position Maus
    }
);
```

Eigenschaften und Methoden des Event-Objekts

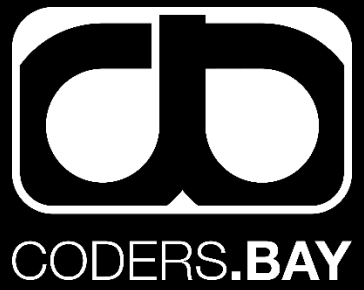
CODERS.BAY

Selektor	Beschreibung
`currentTarget`	Enthält das aktuelle Element während der Bubbling-Phase
`data`	Enthält ein optionales Datenobjekt, welches der Event-Methode übergeben wurde
`delegateTarget`	Enthält das Element, an dem der Event-Listener registriert wurde
`isDefaultPrevented()`	Gibt an, ob `preventDefault()` auf dem Event-Objekt aufgerufen wurde
`isImmediatePropagationStopped()`	Gibt an, ob `stopImmediatePropagation()` auf dem Event-Objekt aufgerufen wurde
`isPropagationStopped()`	Gibt an, ob `stopPropagation()` auf dem Event-Objekt aufgerufen wurde
`metaKey`	Enthält eine Angabe darüber, ob die sogenannte *Meta*-Taste gedrückt wurde (im Falle von Mac-Tastaturen die Command-Taste, im Fall von Windows-Tastaturen die Windows-Taste), während das Event ausgelöst wurde
`namespace`	Enthält den Namensraum des Events
`pageX`	Enthält die Mausposition relativ zum linken Rand des Dokuments
`pageY`	Enthält die Mausoption relativ zum oberen Rand des Dokuments
`preventDefault()`	Verhindert das Ausführen der Standardaktion für ein Event

Eigenschaften und Methoden des Event-Objekts

CODERS.BAY

Selektor	Beschreibung
`relatedTarget`	Enthält zu einem Element, für das ein Event ausgelöst wurde, das Element, welches in einem direkten Zusammenhang zu dem Event steht (beispielsweise im Falle eines mouseout-Events das Element, auf dem durch die gleiche Nutzeraktion das mouseover-Event ausgelöst wurde).
`result`	Enthält das Ergebnis eines vorher für ein Event ausgelösten Event-Listeners
`stopImmediatePropagation()`	Verhindert sofort das weitere Aufsteigen des Events während der Bubbling-Phase
`stopPropagation()`	Verhindert das weitere Aufsteigen des Events während der Bubbling-Phase
`target`	Enthält das Element, welches das Event ausgelöst hat
`timeStamp`	Enthält einen Zeitstempel, der den Zeitpunkt angibt, an dem das Event ausgelöst wurde
`type`	Enthält den Type des Events
`which`	Enthält im Falle von Maus- oder Tastatur-Events die Maustaste bzw. Taste auf der Tastatur, die gedrückt wurde



Zusammenfassung

Zusammenfassung

CODERS.BAY

- jQuery ist eine Bibliothek, die vor allem browserspezifische Details verbirgt und Helfermethoden für wiederkehrende Aufgaben anbietet, die sich browserübergreifend einsetzen lassen.
- Den zentralen Dreh- und Angelpunkt für das Arbeiten mit jQuery bildet die Methode `jQuery()` bzw. `$().`
- Als Argumente kann man dieser Methode u. a. einen Selektor, ein bestehendes Element oder eine HTML-Zeichenfolge übergeben
- Als Rückgabewert liefert die Methode ein Wrapperobjekt (*jQuery-Objekt*), welches die entsprechenden Elemente um zusätzliche Methoden "erweitert" (*jQuery-Methoden*).
- So bietet ein jQuery-Objekt verschiedene Methoden für das Arbeiten mit dem DOM, dazu zählen:
  - Methoden, um auf den Inhalt von Elementen zuzugreifen
  - Methoden, um auf ausgewählte Elemente zu filtern
  - Methoden, um auf Attribute zuzugreifen
  - Methoden, um auf CSS-Eigenschaften zuzugreifen
  - Methoden, um zwischen Elementen zu navigieren
  - Methoden, um Elemente bzw. ihre CSS-Eigenschaften zu animieren
- Für das Arbeiten mit Events stellt jQuery verschiedene Methoden zur Verfügung, um Event-Listener zu registrieren, u. a.:
  - Methoden, um Event-Listener für allgemeine Ereignisse zu registrieren
  - Methoden, um Event-Listener für Mausereignisse zu registrieren
  - Methoden, um Event-Listener für Tastaturereignisse zu registrieren
  - Methoden, um Event-Listener für Formularereignisse zu registrieren

Zusammenfassung

CODERS.BAY

- Auch das Erstellen von Ajax-Anfragen gestaltet sich durch eine Reihe von Helfermethoden einfacher, u. a.:
  - eine Methode um beliebige Ajax-Anfragen zu erstellen
  - eine Methode, um `GET`-Anfragen zu erstellen
  - eine Methode, um `POST`-Anfragen zu erstellen
  - eine Methode, um HTML-Inhalt per Ajax direkt in ein Element zu laden
  - eine Methode, um JavaScript-Dateien zu laden
  - eine Methode, um JSON-Dateien zu laden
- Die meisten Helfermethoden lassen sich dabei zu verschiedenen Zwecken einsetzen, beispielsweise können über die Methode `attr()` HTML-Attribute sowohl gelesen als auch geschrieben, über die Methode `css()` CSS-Eigenschaften gelesen und geschrieben und über die Event-Methode Event-Listener registriert oder wieder entfernt werden

Vergleich zwischen jQuery und purem JavaScript

jQuery ist weitestgehend browserunabhängig, die Varianten mit purem JavaScript nicht immer!

CODERS.BAY

Arbeiten mit dem DOM	jQuery	Pures JavaScript

CSS-Klasse hinzufügen	`$(element).addClass(`
	`  newClassName`
	`);`
		`if (element.classLIst) {`
		`  element.classList.add(newClassName);`
		`} else {`
		`  element.className += ' ' + newClassName,`
		`}`

Auf Kindelemente zugreifen	`$(element).children();`	`element.children`

über Elemente iterieren	`$(selector).each(`
	`  (index, element) => {`
	`  }`
	`);`
		`let elements = document.querySelectorAll(`
		`  selector`
		`);`
		`Array.prototype.forEach.call(`
		`  elements, (element, index) => {`
		`  }`
		`);`

Elemente unterhalb eines Elements suchen	`$(element).find(`
	`  selector`
	`);`
		`element.querySelectorAll(`
		`  selector`
		`);`

Elemente suchen	`$(selector);`	`document.querySelectorAll(selector);`
auf Attribute zugreifen	`$(element)attr(name);`	`element.getAttribute(name);`

Vergleich zwischen jQuery und purem JavaScript

CODERS.BAY

Arbeiten mit dem DOM	jQuery	Pures JavaScript
HTML-Inhalte lesen	`$(element).html();`	`element.innerHTML;`
HTML-Inhalte schreiben	`$(element).html(content);`	`element.innerHTML = content;`
Textinhalte lesen	`$(element).text();`	`element.textContent;`
Textinhalte schreiben	`$(element).text(content);`	`element.textContent = content;`
nächstes Element	`$(element).next();`	`element.nextElementSibling;`
vorheriges Element	`$(element).prev()`	`element.previousElementSibling;`
Arbeiten mit Ereignissen		

Event-Listener hinzufügen	`$(element).on(`
	`  eventName,`
	`  eventHandler`
	`);`
		`element.addEventListener(`
		`  eventName,`
		`  eventHandler`
		`);`

Event-Listener entfernen	`$(element).off(`
	`  eventName,`
	`  eventHandler`
	`);`
		`element.removeEventListener(`
		`  eventName,`
		`  eventHandler`
		`);`

Vergleich zwischen jQuery und purem JavaScript

CODERS.BAY

Arbeiten mit Ereignissen	jQuery	Pures JavaScript
Funktion bei Laden des Dokuments ausführen	`$(document).ready(() => { // });`	`function ready(callback) { if (document.readyState != 'loading') { callback(); } else { document.addEventListener( 'DOMContentLoaded', callback ); } }`
Arbeiten mit Ajax-Anfragen		
GET-Anfrage senden	`$.ajax({ type: 'GET', url: 'url', success: (response) => {} , error: () => {} });`	`let request = new XMLHttpRequest(); request.open('GET', 'url', true); request.onload = () => { if (request.status >= 200 && request.status < 400) { let response = request.responseText; } else { } }; request.onerror = () => {}; request.send();`

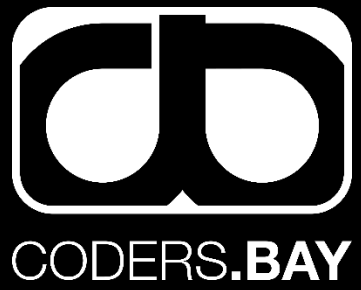
Vergleich zwischen jQuery und purem JavaScript

CODERS.BAY

Arbeiten mit Ajax-Anfragen	jQuery	Pures JavaScript

POST-Anfrage senden	`$.ajax({`
	`  type: 'POST',`
	`  url: 'url',`
	`  data: data`
	`});`
		`let request = new XMLHttpRequest();`
		`request.open('POST', 'url', true);`
		`request.setRequestHeader(`
		`  'Content-Type',`
		`  'application/x-www-form-urlencoded; ' +`
		`  'charset=UTF-8);`
		`request.send(data);`

JSON per Ajax laden	`$.getJSON(`
	`  'data.json',`
	`  (data) => {}`
	`);`
		`let request = new XMLHttpRequest();`
		`request.open('GET', 'data.json', true);`
		`request.onload = () => {`
		`  if (request.status >= 200 && request.status < 400) {`
		`  let data = JSON.parse(request.responseText);`
		`  } else {`
		`  }`
		`};`
		`request.onerror = () => {`
		`};`
		`request.send();`



Quelle

Quelle: JavaScript - Das umfassende Handbuch

Rheinwerk Computing
ISBN 978-3-8362-3838-0