JS-Formulare

Lernziele

- Wissen, wie man das Standardverhalten des Formular-Submits mit `.preventDefault()` verhindert
- Wissen, wie man auf `submit`-Ereignisse hört: das `event`-Objekt und seine `target`-Eigenschaft
- Eingabewerte lesen:
  - `event.target.elements`
  - `FormData`
  - die Rolle der `name`-Attribute für Formularfelder
- Das `input`-Ereignis verstehen
- Wissen, wie man ein Eingabefeld programmatisch fokussiert
- Wissen, wie man ein Formular zurücksetzt

Auf das `submit`-Ereignis hören und das Standardverhalten verhindern

Um das Verhalten des `submit`-Ereignisses zu verhindern, müssen Sie

- das `event`-Objekt als Argument der Event-Listener-Arrow-Funktion empfangen
- `event.preventDefault()` aufrufen

```
const form = document.querySelector('[data-js="form"]');

form.addEventListener("submit", (event) => {
  event.preventDefault();
});
```

Durch den Aufruf(calling) von `event.preventDefault()` führt der Browser keine GET-Anfrage durch, die beim Absenden die Seite neu laden würde.

Das `event`-Objekt und `event.target``

Das `event`-Objekt wird jedes Mal erstellt, wenn ein Ereignis ausgelöst wird. Sie können es als ersten Parameter in der Callback-Funktion akzeptieren und somit im Funktionskörper darauf zugreifen (z. B. über `event.preventDefault()`).

Für den Moment ist die wichtigste Methode des `event`-Objekts `.preventDefault()`.

`event.target` ist ein Verweis auf das Element, von dem das Ereignis ausgegangen ist - in diesem Fall das Formular.

```
form.addEventListener("submit", (event) => {
  event.preventDefault();
```

```
  console.log(event.target);
});
// Ausgabe:
// <form data-js="form">
//      <fieldset>...</fieldset>
//      ...
//      <button type="submit">Submit</button>
//  </form>
```

Zugriff auf interaktive Felder: `event.target.elements` und das `name`-Attribut

Während `event.target` das gesamte Formular darstellt, ist `event.target.elements` eine Sammlung aller Formularelemente (form fields, field sets und buttons).

Sie erhalten Zugriff auf ein bestimmtes Formularfeld über sein `name`-Attribut und die Punktnotation:

```
form.addEventListener("submit", (event) => {
  event.preventDefault();

  const formElements = event.target.elements;

  console.log(formElements.firstName);
  console.log(formElements.firstName.value);
});
```

Beachten Sie, dass

- `event.target.elements` in der Variablen `formElements` gespeichert wird, um die Lesbarkeit zu verbessern,
- `firstName` der String-Wert des entsprechenden `name`-Attributs ist, wie in `<input name="firstName"/>`, und `firstName.value` die Benutzereingabe für das Feld mit `name="firstName"` zurückgibt.

Verwenden von Eingabewerten

Sie können alle Eingabewerte des Formulars mit `FormData()` abrufen. Dieser Konstruktor verwendet `event.target` und kann anschließend in ein verwendbares Objekt umgewandelt werden:

```
form.addEventListener("submit", (event) => {
  event.preventDefault();

  const formData = new FormData(event.target);
  const data = Object.fromEntries(formData);

  console.log(data);
});
```

Dies ist sehr nützlich, um einfach auf die Eingabedaten eines gesamten Formulars zuzugreifen.

> 💡 Obwohl die Verwendung von `FormData` viel weniger ausführlich ist, ist `event.target.elements` sehr nützlich, wenn Sie auf einzelne Formularfelder zugreifen möchten. (Spoiler-Alarm: Zum Beispiel, wenn Sie nach dem Zurücksetzen des Formulars ein bestimmtes Feld fokussieren möchten.)

Ausnahme: Lesen von Werten aus Checkboxen

Checkboxen haben zwei Zustände: angekreuzt ("true") und nicht angekreuzt ("false"). Im Gegensatz zu anderen Eingabetypen spiegelt das `value`-Attribut diese Änderung nicht wider, sondern wird nur als Bezeichner für die Checkbox verwendet.

Sie können den Status der Checkbox stattdessen über die `.checked-Eigenschaft` abrufen.

Stellen Sie sich die folgende Checkbox vor

```
<input type="checkbox" name="colorBlue" value="blue" data-js="blue" />
```

und das entsprechende JavaScript:

```
console.log(formElements.colorBlue.checked); // Ausgabe: true oder false
console.log(formElements.colorBlue.value); // Ausgabe (immer): blue
```

Sie können auch auf jedes An- oder Abhaken der Checkbox reagieren:

```
const checkbox = document.querySelector('[data-js="blue"]');

checkbox.addEventListener("input", (event) => {
  console.log(event.target.checked); // Ausgabe: true oder false
});
```

Das input-Ereignis(event)

Gelegentlich möchten Sie vielleicht etwas tun, wenn sich der Wert eines einzelnen Feldes ändert, noch bevor das Formular abgeschickt wird.

Das `input`-Ereignis wird jedes Mal ausgelöst, wenn sich der Wert eines Formularfeldes ändert. Zum Beispiel wird ein `<textarea />` dieses Ereignis bei jedem Tastendruck auslösen.

```
const messageInput = document.querySelector('[data-js="message"]');

messageInput.addEventListener("input", (event) => {
```

```
  console.log(event.target.value);
});
```

> ! Verwechseln Sie das `input`-Ereignis nicht mit dem `change`-Ereignis, das nur ausgelöst wird, nachdem der Inhalt eines Feldes vom Benutzer durch Drücken der Eingabetaste oder Verschieben des Fokus auf das nächste Feld bestätigt wurde.

Eingabefelder fokussieren

Sie können ein Eingabefeld mit der `.focus()`-Methode fokussieren. Dies kann verwendet werden, um die Benutzererfahrung nach dem Abschicken eines Formulars zu verbessern.

```
form.addEventListener("submit", (event) => {
  event.preventDefault();
  // [...] Formulardaten verarbeiten
  event.target.elements.message.focus();
});
```

Dies wird ein Formularfeld mit dem Attribut `name="message"` fokussieren.

Formulare zurücksetzen

Sie können alle Formularfelder mit der `.reset()`-Methode auf ihren Standardwert zurücksetzen.

```
form.addEventListener("submit", (event) => {
  event.preventDefault();
  // [...] Formulardaten verarbeiten
  event.target.reset();
});
```

Dies ist oft praktisch in Kombination mit `.focus()`. Denken Sie an einen Chat: Nachdem die Nachricht gesendet wurde, wird das Eingabefeld gelöscht und erneut fokussiert, sodass Benutzer die nächste Nachricht schreiben können.

Resourcen

- Event interface