React Effekte und Fetch

Lernziele

- [ ] Wissen, dass `fetch` ein Nebeneffekt ist
- [ ] Wissen, wie man den `useEffect` Hook verwendet
- [ ] Verstehen der Callback-Funktion von `useEffect`
- [ ] Verstehen des Abhängigkeitsarrays von `useEffect`
- [ ] Verstehen der Aufräumfunktion von `useEffect`
- [ ] Kennen anderer Nebeneffekte und Fälle für `useEffect`

Effekte in React

Effekte sind eine Möglichkeit, React-Komponenten mit externen Systemen zu synchronisieren.

Beispiele für Interaktionen mit externen Systemen umfassen:

- Direktes Manipulieren des DOM (z. B. Festlegen des Dokumenttitels)
- Durchführen von Netzwerkanfragen, um Daten abzurufen
- Arbeiten mit anderen Web-APIs
- Einrichten und Entfernen von Abonnements und globalen Event-Handlern
- Einstellen von Timern
- Integrieren von Drittanbieter-Bibliotheken

Die Verwendung eines Effekts ist ein Ausweg aus der deklarativen Welt von React. Es ist eine Möglichkeit, imperativen Code auszuführen, der nicht direkt mit dem Rendering der Benutzeroberfläche zusammenhängt. Während die Komponentenfunktion rein sein muss, sind Effektfunktionen von Natur aus nicht rein. Sie kapseln Nebeneffekte.

Ein Effekt wird als eine Funktion definiert, die nach dem Rendern der Komponente (und dem Aktualisieren des DOM) ausgeführt wird. Er kann so synchronisiert werden, dass er nicht nur beim Einbinden, sondern auch dann ausgeführt wird, wenn sich alle oder nur bestimmte reaktive Werte innerhalb der Komponentenfunktion geändert haben.

Effektfunktionen können eine Aufräumfunktion zurückgeben, die ausgeführt wird, bevor die Effektfunktion erneut ausgeführt wird, oder wenn die Komponente entfernt wird.

> 💡 Ein **reaktiver Wert** ist ein Wert, der sich ändert: Props, State, deren Ableitungen oder Werte und Funktionen, die innerhalb einer Komponentenfunktion deklariert werden.

> 💡 **Mounting** bedeutet, dass eine Komponente gerendert, in das DOM eingefügt und zum ersten Mal auf dem Bildschirm angezeigt wurde. Danach können verschiedene Updates und erneute Renderings auftreten (z. B. aufgrund von Zustandsänderungen). **Unmounting** bedeutet, dass die Komponente entfernt wird und nicht mehr auf dem Bildschirm angezeigt wird.

`useEffect`

Der `useEffect` Hook wird verwendet, um Effekte zu einer React-Komponente hinzuzufügen. Er nimmt zwei Argumente entgegen:

- eine Funktion, die den Effekt definiert (normalerweise eine anonyme Funktion)
- ein Array von Variablen, von denen der Effekt abhängt

Zum Beispiel wird der folgende Code den Titel der Komponente auf den Wert der `title`-Prop aktualisieren:

```
import { useEffect } from "react";

function Title({ title }) {
  useEffect(() => {
    // Das Aktualisieren des Dokumenttitels ist ein Nebeneffekt,
    // der nicht direkt mit dem Rendering der UI zusammenhängt
    document.title = title;
  });

  return <h1>{title}</h1>;
}
```

Effektabhängigkeiten

Der obige Effekt wird ausgeführt, nachdem die Komponente gerendert und das DOM aktualisiert wurde. Aber das ist viel häufiger als nötig. Der Effekt sollte nur ausgeführt werden, wenn sich die `title`-Prop ändert. Um dies zu erreichen, können wir ein Array reaktiver Werte an den `useEffect()` Hook übergeben. Der Effekt wird nur ausgeführt, wenn sich einer der reaktiven Werte im Array ändert.

```
import { useEffect } from "react";

function Title({ title }) {
  useEffect(() => {
    document.title = title;
  }, [title]);

  return <h1>{title}</h1>;
}
```

Dies wird wichtig, wenn die Komponentenfunktion mehr als eine Prop oder State-Variable hat. Stellen Sie sich vor, es gibt eine `count`-State in der Komponente:

```
import { useEffect, useState } from "react";

function Title({ title }) {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = title;
```

```
  }, [title]);

  return (
    <div>
      <h1>{title}</h1>
      <p>{count}</p>
      <button type="button" onClick={() => setCount(count + 1)}>
        Increment
      </button>
    </div>
  );
}
```

Die Effektfunktion wird nur ausgeführt, wenn die `title`-Variable anders ist als der vorherige Wert. Die `count`-State-Variable ist nicht Teil des Arrays, daher wird der Effekt nicht ausgeführt, wenn sich der `count`-State ändert.

> 💡 Fügen Sie immer alle reaktiven Werte, die in der Effektfunktion verwendet werden, zum Abhängigkeitsarray hinzu. React wird mit ESLint-Regeln geliefert, die Sie warnen, wenn Sie vergessen, eine Variable zum Abhängigkeitsarray hinzuzufügen.

> **Wenn der Effekt keine Abhängigkeiten hat, sollte das Abhängigkeitsarray leer sein: `[]`.**

Ein leeres Abhängigkeitsarray sagt React, diesen Effekt nur einmal auszuführen: Wenn die Komponente zum ersten Mal auf dem Bildschirm erscheint.

Aufräumfunktion

Die Effektfunktion kann eine Aufräumfunktion zurückgeben, die ausgeführt wird, bevor die Effektfunktion erneut ausgeführt wird, oder wenn die Komponente entfernt wird.

```
import { useEffect } from "react";

function Title({ title }) {
  useEffect(() => {
    // eine Kopie des alten Titels machen
    const oldTitle = document.title;

    document.title = title;

    // Aufräumfunktion
    return () => {
      // rückgängig machen, was wir getan haben, indem wir den alten Titel
wieder einstellen
      document.title = oldTitle;
    };
  }, [title]);

  return <h1>{title}</h1>;
}
```

Die Aufräumfunktion sollte die Nebeneffekte der Effektfunktion rückgängig machen. Im obigen Beispiel setzt die Aufräumfunktion den Dokumenttitel auf den Standardwert zurück.

Wenn die Effektfunktion verwendet wird, um ein Abonnement oder einen globalen Event-Handler einzurichten, sollte die Aufräumfunktion das Abonnement oder den Event-Handler entfernen.

```
import { useEffect, useState } from "react";

function WindowWidth() {
  const [windowWidth, setWindowWidth] = useState();

  useEffect(() => {
    function handleResize(event) {
      setWindowWidth(event.target.innerWidth);
    }

    window.addEventListener("resize", handleResize);

    // Aufräumfunktion
    return () => {
      window.removeEventListener("resize", handleResize);
    };
  }, []);

  return <p>Das Fenster ist {windowWidth}px breit. 🪚</p>;
}
```

Bei Timern sollte die Aufräumfunktion den Timer löschen.

```
import { useEffect, useState } from "react";

function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const timer = setInterval(() => {
      setSeconds((s) => s + 1);
    }, 1000);

    // Aufräumfunktion
    return () => {
      clearInterval(timer);
    };
  }, []);

  return <p>Der Timer steht bei {seconds} Sekunden. ⏱️</p>;
}
```

> 📙 Lesen Sie mehr über **Synchronizing with Effects** in den React-Dokumenten.

> 📙 Auch wenn Effekte unglaublich nützlich sein können, benötigen Sie möglicherweise tatsächlich keinen Effekt, um das zu tun, was Sie wollen. Lesen Sie mehr über **You might not need an effect** in den React-Dokumenten.

Wie man Daten in React abruft

Eine der häufigsten Anwendungsfälle für Effekte ist das Abrufen von Daten von einer externen API.

Das Konzept ist wie folgt: Nachdem die Komponente zum ersten Mal gerendert wurde, wird eine Effektfunktion ausgeführt und Daten von einer externen API abgerufen. Sobald die Daten abgerufen wurden, wird eine State-Variable innerhalb der Effektfunktion gesetzt. Wenn das Fetching von einer Prop oder State-Variable abhängig ist, wird die Effektfunktion erneut ausgeführt, wenn sich die Variable ändert.

Die Effektfunktion selbst kann nicht asynchron sein, aber sie kann asynchrone Funktionen aufrufen. Um dies zu umgehen, können Sie eine asynchrone Funktion innerhalb der Effektfunktion definieren und sie sofort aufrufen (ohne das Ergebnis tatsächlich abzuwarten).

Das folgende Beispiel zeigt, wie man Daten von einer API abruft und die Daten in einer Komponente anzeigt:

```
import { useEffect, useState } from "react";

function Jokes() {
  const [jokes, setJokes] = useState([]);

  useEffect(() => {
    async function startFetching() {
      const response = await fetch(
        "https://example-apis.vercel.app/api/bad-jokes"
      );
      const jokes = await response.json();

      setJokes(jokes);
    }

    startFetching();
  }, []);

  return (
    <ul>
      {jokes.map(({ id, joke }) => (
        <li key={id}>{joke}</li>
      ))}
    </ul>
  );
}
```

Wenn die Daten, die Sie abrufen möchten, von einer Prop oder State-Variable abhängen, müssen Sie diese zum Array der Variablen hinzufügen, von denen der Effekt abhängt:

```jsx
import { useEffect, useState } from "react";

function Joke({ id }) {
  const [joke, setJoke] = useState();

  useEffect(() => {
    async function startFetching() {
      const response = await fetch(
        `https://example-apis.vercel.app/api/bad-jokes/${id}`
      );
      const joke = await response.json();

      setJoke(joke);
    }

    startFetching();
  }, [id]);

  if (!joke) {
    return null;
  }

  return <h2>{joke.joke}</h2>;
}
```

Der obige Ansatz funktioniert gut genug für einfache Anwendungsfälle. Es werden jedoch einige wichtige Funktionen nicht abgedeckt:

- Er behandelt keine Race Conditions. (Wenn der Benutzer die `id`-Prop ändert, bevor der erste Fetch abgeschlossen ist, besteht die Möglichkeit, dass der falsche Witz angezeigt wird.)
- Er behandelt keine Ladezustände.
- Er behandelt keine Fehler. (Weder Netzwerkfehler noch Fehler von der API.)
- Er behandelt kein Caching.

In Zukunft werden wir eine Datenabrufbibliothek verwenden, um diese Probleme zu lösen.

> 💡 Auch wenn Sie eine Datenabrufbibliothek verwenden, wird die Bibliothek Effekte (und den `useEffect` Hook) im Hintergrund verwenden, um Daten abzurufen.

📙 Lesen Sie mehr über **Fetching Data** in den React-Dokumenten. Die Dokumentation beschreibt auch eine Möglichkeit, Race Conditions zu behandeln.

Resources

- React docs: Synchronizing with Effects
- React docs: Fetching data example with useEffect
- React docs: You Might Not Need an Effect