Javascript

Hendelser og callbacks

Bjarte Kileng

HVL

9. november, 2021

Brukerhandlinger og hendelser

- Nettleser kan reagere på handlinger til bruker.
  - Bruker klikker på en knapp.
  - Musen beveges over et web-element.
  - Bruker fyller inn tekst i et input-element.
  - Web-dokumentet er ferdig lastet inn i nettleser.
- En hendelse (DOM event) er resultat av bruker sine handlinger.
  - Bruker som klikker på knapp resulterer i en **click** hendelse på knappen.
- Nettleser kan kjøre JavaScript-kode når hendelse skjer.
- For å reagere på en hendelse må nettleser vite:
  - Hvilket elementet skal reagere på hendelsen (f.eks. et *button* element)?
  - Hvilken hendelsen skal det reageres på (f.eks. **click**)?
  - Hvilken JavaScript funksjon som skal kjøres når hendelsen skjer?

addEventListener

- Metoden *addEventListener* brukes for å legge på lytter for hendelse.
- Eksempel med *addEventListener*:

```
const root = document.getElementById("root");
const buttonRef = root.querySelector("button[type='button']");

function helloWorld () {
    console.log("Hello world");
}

buttonRef.addEventListener("click",helloWorld);
```

- Andre argument til *addEventListener* er en funksjon:
  - Funksjonen blir satt på vent og kjøres først når hendelsen skjer.
  - En funksjon gitt som argument kalles for en *callback*.
- Kan legge på flere lyttere for samme hendelse på samme element.
- Finnes eldre mekanismer for hendelser, ikke pensum i DAT108.

Callbacks

- En *callback* er en funksjon gitt som parameter ved et funskjonskall:

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

- Kjøre funksjonen *run* med funksjon som argument (callback):

```
run(f);
```

- Kjører *f* og kjører *run* på resultatet av *f* (ikke en callback):

```
run(f());
```

Callback og funksjonskall

Det er funksjonen selv som gis som argument.

Argumentet er ikke resultatet etter å ha kjørt funksjonen.

Aldri parenteser «()» bak callback som parameter

Bruk av callbacks

- Callbacks kan nøstes i mange nivåer og gi uoversiktlig kode:

```
document.addEventListener("DOMContentLoaded",
    () => {
        const rootElm = document.getElementById("root");
        rootElm.querySelector("button[type='button']").addEventListener("click",
            (event) => {
                window.setInterval(
                    () => {
                        event.target.classList.toggle("gyldig");
                    }
                ,2000);
            }
        );
    }
);
```

- Slik bruk av callbacks kalles ofte for *callback hell*.

Kode med callbacks

- Gi navn til callback-funksjonen, og flat ut kall-sekvensen.
- Bruk JSDoc og dokumenter bruken av callbacks.
- Bruk **Promise** hvis mulig (ikke pensum i DAT108).
- Bruk `async` and `wait` sammen med **Promise** (ikke pensum).

Flate ut kall-sekvensen

```
function run(event) {
    window.setInterval(
        () => {event.target.classList.toggle("gyldig")}
        ,2000
    );
}

function init() {
    const rootElm = document.getElementById("root");
    const button = rootElm.querySelector("button[type='button']")
    button.addEventListener("click",run);
}

document.addEventListener("DOMContentLoaded",init);
```

Event objektet

- Ved hendelse opprettes et **Event** objekt for hendelsen.
- Eksempler på informasjon fra **Event** objektet:
  - HTML-element som er mål for hendelsen.
  - Posisjon på skjermen der hendelsen skjedde (x,y posisjon).
  - Posisjon i nettleser er hendelsen skjedde (x,y posisjon).
  - Tidspunkt for hendelsen.
  - Taster og knapper som var trykket inn ved hendelsen.
- Objektet gis som parameter til callback-funksjonen.

```
elmRef.addEventListener(hendelse,callback);

function callback(event) {
    console.log(event.target.tagName);
}
```

Callbacks, *this* og *bind*

- Når en hendelseshåndterer kjøres vil normalt *this* være elementet som trigget hendelsen.
  - Skjer også om det er en medlemsmetode som kjøres.
- Funksjoner sin metode *bind* kan angi verdien til *this*.
  - Bruk av *bind* gir ny funksjon der *this* har den oppgitte verdien.

```
this.metode = this.metode.bind(this);
```

- Bruk *bind* på medlemsmetode som er callback ved hendelse.
  - I eksempelet må *metode* bindes til **objekt** med *bind* før bruk.

```
elmRef.addEventListener(hendelse,objekt.metode);
```

Omsluttende funksjon for parametre til callback

- Kan gi parameter til callback ved å omslutte den i en ytre funksjon.

```
elmRef.addEventListener(
    'click',
    (event)=>{this.metode(event,parameter)}
);
```

- Bruk pil-notasjon for omsluttende funksjon.
  - Verdi av *this* betemmes av kontekst der funksjonen deklareres.
- Ved *function* bindes parametre som *this* og *arguments*.
  - Verdi av *this* betemmes når funksjonen kjøres.
  - Kan løses ved å mellomlagre verdien av *this*.

Noen eksempler på hendelser

click:	Klikk med musen.
dblclick:	Dobbeltklikk med musen.
mouseover:	Musen beveges inn i element.
mouseout:	Musen beveges ut av element.
DOMContentLoaded:	Dokumentetstrukturen er ferdig bygget.
load:	Dokument eller ressurs er ferdig lastet.
submit:	Form-skjema submit.
change:	Endring av data i form-felt.
	▶ Når endring er ferdig.
input:	Endring av data i form-felt.
	▶ Under endring, f.eks. når bruker skriver.
invalid:	Feil ved validering, f.eks. *input* med *pattern*.
error:	Feil, f.eks. i JavaScript eller nettforbindelse.

Standardoppførsel ved hendelse

- Klikke på lenke vil navigere til URL gitt i *href* attributtet.
- Klikk på submit-knapp vil sende data til tjener.
- Klikk på reset-knapp vil nullstille form-skjema.
- Feilmeldinger vil logges i konsollet.

Rekkefølge ved hendelse

Alle hendelseshåndterere lagt til med JavaScript skjer først.

Først når alle disse er ferdig med å kjøre blir standardoppførsel utført.

Kansellere standardoppførsel ved hendelse

- **Event** klassen sin metode *preventDefault()* kansellerer standardoppførsel til hendelse.
- Kansellere navigasjon for alle lenker på webside:

```
function stoppNavigasjon(aRef) {
    aRef.addEventListener("click", (event) => {event.preventDefault()});
}

document.querySelectorAll("a").forEach(stoppNavigasjon);
```