React events & state

Vefforritun

Ólafur Sverrir Kjartansson, osk@hi.is

Atburðir í React

- Atburðir í React svipa til DOM atburða og nota þá „undir húddinu“
- Festum *beint á element* í JSX 🙀
  - Eitthvað sem við lærðum í vefforritun 1 að gera **alls ekki**
- camelCase og föll en ekki strengir
  - `<button onClick={activateLasers}>` ekki `<button onclick="activateLasers()">`

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }

  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

Að taka þátt í atburðum

- Í react þurfum við yfirleitt ekki að nota `addEventListener`
- React notar *synthetic events*, ekki *alvöru* event frá browser
- Skráum handler beint í JSX og react sér um

Synthetic event

- React útfærir eigin events út frá W3C spec
  - Getum notað `e.stopPropagation()` og `e.preventDefault()`
- Endurnýtir events og gerir ýmislegt til að fá betra performance
- React: SyntheticEvent

Binding

- Það er algengt að nota class method þegar við skráum handler á class
- Verðum að passa að *binda* rétt til þess að `this` virki
- Nokkrar leiðir í boði

Binding leiðir

- Binda í `constructor`
  - `this.handleClick = this.handleClick.bind(this);`
  - Frekar fyrirferðarmikið

- Binda í event
  - `<button onClick={(e) => this.handleClick(e)}>`
  - Getur orðið fyrirferðarmikið og flókið að lesa
  - Veldur því að fyrir hvert `render` er búið til nýtt fall

- Nota public class field (ekki orðið staðlað) * `handleClick = () => { /* ... */ }`
  - Þægileg leið sem er studd af CRA
  - Mæli með að nota þessa

```
class MyComponent extends Component {
  handleClick(e) {
    console.log(this, e);
  }

  render() {
    return (
      <p>
        <button onClick={this.handleClick}>
          click
        </button>
```

```
      </p>
    );
  }
}
```

Binding í functional components

- Þurfum ekki að pæla í binding í functional components
- Búum til lokun (closure) yfir component sem geymir vísun í skilgreind föll

---

```
function MyComponent() {
  const handleClick = (e) => {
    console.log(e);
  };

  // eða
  function handleClickk(e) {
    console.log(e);
  }

  return (
    <p>
      <button onClick={this.handleClick}>
        click
      </button>
    </p>
  );
}
```

Argument í event handler

- Til að senda gögn í event handler getum við notað
  - `<button onClick={(e) => this.del(id, e)}>Delete</button>`
  - `<button onClick={this.del.bind(this, id)}>Delete</button>`

- Eða við getum útbúið handler sem skilar *öðrum* handler

```
const deleteThing = (id) => (e) => {
  e.preventDefault();
  // Lokun (closure) yfir id
  // id er aðgengilegt þegar
  // atburður á sér stað
};

// ...

const button = (<button onClick={this.deleteThing(1)} />);
```

State

- Staða fyrir component, aðeins sýnileg innan component
- Svipar til `props` en er ekki read-only og breytist með notkun á component
- Stjórnað af component að öllu leiti
- Til fyrir class component og functional components
- Ættum aðeins að setja hluti sem verða birtir í `state`, þ.e.a.s. part af viðmóti

---

Dæmi um state

- Er búið að smella á takka sem sýnir auka upplýsingar?
- Er verið að sækja gögn?
- Hvaða gögn er búið að sækja og birta?
- Hvað er búið að skrifa inn í `<input>`?

State í class components

Byrjunarstaða

- Setjum `this.state = { ... }` í `contstructor`
- Ef við höfum stuðning við member breytur sem `state = { ... }` í class (virkar í CRA)
- **Eina skiptið** sem við setjum `state` beint

```
class MyComponent extends Component {
  constructor(props) {
    super(props);
    this.state = { /* ... */ };
  }
}
```

```
class MyComponent extends Component {
  // Stuðningur við member breytu
  state = { /* ... */ };
}
```

Uppfærsla á `state` í class compoment

- Uppfærsla á `state` fer alltaf fram í gegnum `this.setState()`
- Uppfærslur eru keyrðar asynchronously, getum ekki treyst á að uppfærsla hafi átt sér stað í næstu línu

---

- Ef við viljum bíða eftir að búið sé að setja `state` getum við notað `this.setState((prevState, props) => { /* ... */ })`
- Getum uppfært einn hlut í stöðu í einu, þurfum ekki að uppfæra allt í einu

---

```
// 👎
this.state.comment = 'Hello';
```

```
// 👍
this.setState({ comment: 'Hello' });
```

```
// 👎
this.setState({
  counter:
    this.state.counter + this.props.incr,
});
```

```
// 👍
this.setState((prevState, props) => ({
```

```
  counter: prevState.counter + props.incr,
}));
```

State í functional components

- Nýlega (frá React 16.8, 2019) hægt að skilgreina state og aðra virkni án þess að nota class components með *hooks*
- Kemur ekki í staðinn fyrir classes
- Getur stytt kóða töluvert og gert (oftast!) læsilegri

- Leyfa okkur að „hooka“ okkur inn í React state og lifecycle methods frá functional componentum
- React skilgreinir fyrir grunnaðgerðir og aðrar sem geta verið nytsamlegar
- Nánar um hooks
  - Using the State Hook

Hooks reglur

Verðum að fylgja relgum um hooks, lendum annars í vandræðum, aðallega:

- Aðeins kalla í hooks yst í falli *Ekki í lykkjum, flæðistýringum eða innriföllum
- Aðeins kalla í hooks úr React componentum *Ekki úr „venjulegum“ JavaScript föllum

---

useState

- `useState` leyfir okkur að nota state í functional components
- Áður þurftum við að breyta þeim í class component **Mikill* auka kóði
- Skilar tvennd í fylki: fyrra er gildið, seinna er fall til að breyta gildi

---

- Sendum inn upphafsgildi þegar við skilgreinum state fyrst
- `const [count, setCount] = useState(0);` *Notar *array destructuring*
- Getum skilgreint eins mörg `useState` og við þurfum í component

```jsx
import React, { useState } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button
        onClick={() => setCount(count + 1)}
      >
        Click me
      </button>
    </div>
  );
}
```

Flæði gagna

- Componentum á að vera alveg sama hvort aðrir componentar hafi stöðu eða ekki; séu functional eða ekki
- `state` er háð útfærslu component og er aldrei aðgengilegt öðrum componentum
  *Nema component sendi part af stöðu áfram í *prop* hjá öðrum component

---

Gögn flæða niður

- Gögn í react flæða alltaf *niður* **top-down* eða *unidirectional* gagnaflæði
- Þetta vefst fyrir mörgum sem byrja að nota react
- Vegna reconciliation útfærslu er ódýrar fyrir okkur að teikna **allt** UI aftur en að breyta ákveðnum hlut
  - Eigum aldrei (mjög sjaldan!) að þurfa að nota sjálf DOM aðgerðir
- Skoðum nánar í næsta fyrirlestri