React実習

04. イベント処理

株式会社ジードライブ

今回学ぶこと

- 前提知識：JavaScriptでのイベント処理
- Reactアプリ内でのイベント処理の実装方法
  - useState, useRefを使用したフォーム処理の実装方法

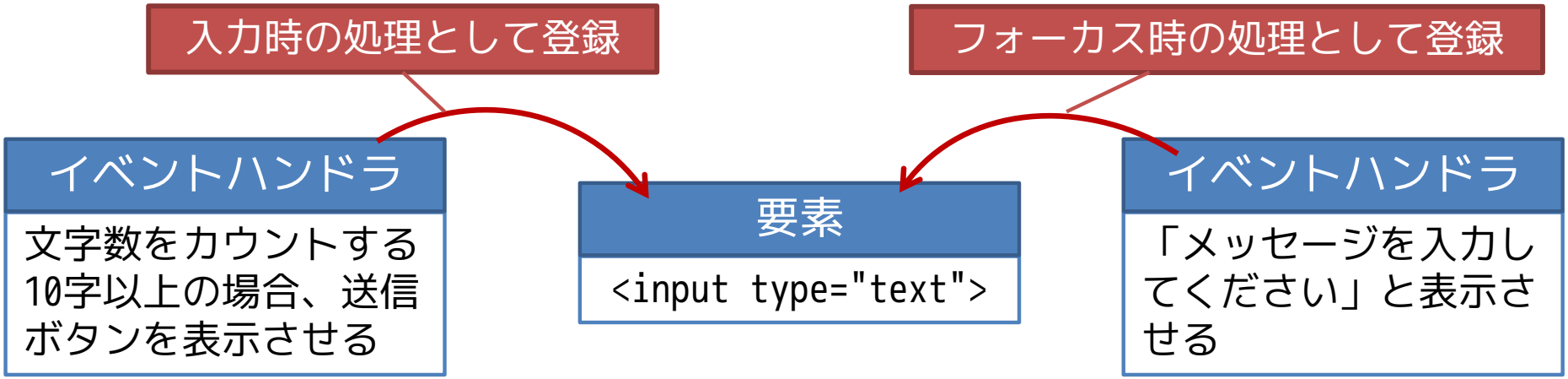
前提知識
JavaScriptでのイベント処理

イベントとは

- マウス操作、キーボード操作、フォーム操作などのブラウザ上で発生する様々な出来事や状態の変化
  - マウスのクリック
  - キーの入力
  - フォームの送信、etc...
- JavaScriptではそのようなイベントの発生時にコードを実行する仕組みが用意されている

イベントハンドラ

- イベントが発生した際の対応処理をイベントハンドラと呼び、関数として定義する
- イベントハンドラ関数は、イベントの種類と共に、要素に紐づけることができる



イベントハンドラの実装方法

- タグに対してイベントハンドラ属性を設定する方法

例：ボタンクリック時に、アラートウィンドウに「100」を表示

```
<button onclick="const x = 100; alert(x);">Click</button>
```

- addEventListener()を使う方法
  - 一般的には、こちらを利用する

例：ボタンクリック時に、アラートウィンドウに「100」を表示

```
<button>Click</button>
<script>
const btn = document.querySelector("button");
btn.addEventListener("click", (e) => {
    const x = 100;
    alert(x);
});
</script>
```

querySelector()はCSSセレクタを使い要素を選択するメソッド。複数の要素が該当する場合は、HTML上に一番最初に現れる要素を選択する

click時の処理を追加

イベントハンドラ属性の例

イベントハンドラ属性	イベント発生タイミング
onclick	マウスをクリックした
onmouseover	マウスカーソルが要素の上に移動した
onmouseout	マウスカーソルが要素から外れた
onchange	フォーム部品の値が変わった
onsubmit	フォームの送信ボタンが押された
onkeydown	要素の上をキーで押した
onload	文書の読み込み完了
onscroll	スクロールバーでスクロールした時

addEventListener

- 要素にイベントハンドラ関数を紐づけるには、下記のメソッドを使用する

要素.addEventListener(イベントの種類, イベントハンドラ関数)

  - イベントの種類には、click, focus, input, … などがある
  - イベントハンドラ関数は無名関数(アロー関数)で記述可能

例: button要素をクリックしたら、「こんにちは」と表示する

```
const btn = document.querySelector("button");
btn.addEventListener("click", () => alert("こんにちは"));
```

イベントの種類

イベントハンドラ(イベント発生時の処理内容)

マウスイベント

- addEventListenerの引数として渡すことができるマウスイベントの種類としては、以下のようなものが存在する

イベント	用途
click	クリックした時の処理の登録
dblclick	ダブルクリックした時の処理の登録
mouseover	マウスカーソルが要素の上に移動した時の処理の登録
mouseout	マウスカーソルが要素から外れた時の処理の登録
mousedown	マウスボタンが下がった時の処理の登録
mouseup	マウスボタンが上がった時の処理の登録
mousemove	マウスカーソルが移動した時の処理の登録

フォームイベント

- addEventListenerの引数として渡すことができるフォームイベントの種類としては、以下のようなものが存在する

イベント	用途
change	テキスト入力が確定した時、ラジオボタン等の選択で変更があった時の処理の登録
input	テキスト入力中の処理の登録
submit	フォーム送信時の処理の登録
focus	フォームの部品にフォーカスが当たった時の処理の登録
blur	フォーカスが外れた時の処理の登録

マウスイベントの例

例：画像にマウスカーソルを乗せると、画像のalt属性値が
h1要素として表示される

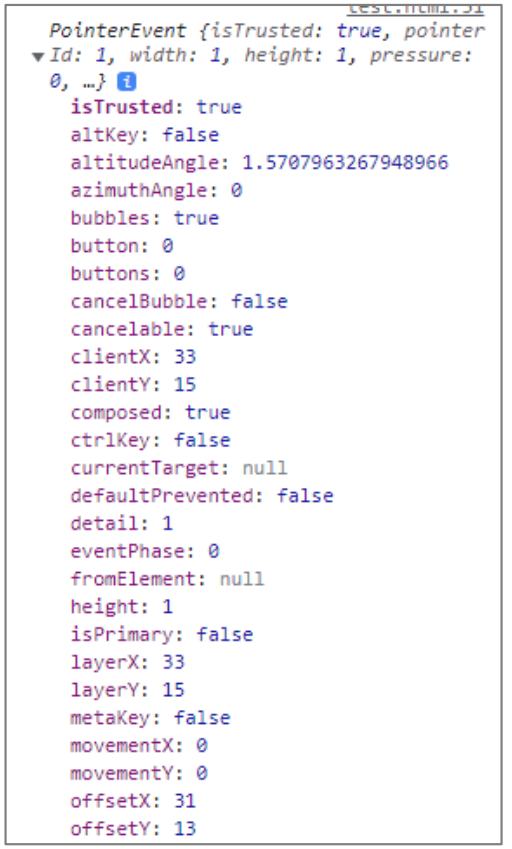
```
<h1>UNTITLED</h1>
<img src="images/cat.jpg" alt="空を見上げる猫" width="300">
<img src="images/dog.jpg" alt="尻尾を振る子犬" width="300">

<script>
const h1 = document.querySelector('h1');
document.querySelectorAll('img')
.forEach(img => {
  img.addEventListener('mouseover', (event) => {
    h1.textContent = event.target.getAttribute('alt');
  });

  img.addEventListener('mouseout', (event) => {
    h1.textContent = 'UNTITLED'
  });
});
</script>
```

Eventオブジェクト

- イベントハンドラ関数では、引数を設定できる
  - この引数はEvent型のオブジェクトで、イベントの種類やそのイベントが発生した要素の情報などを保持している

```
const btn = document.querySelector("button");
btn.addEventListener("click", (event) => {
  console.log(event);
});
```

任意の引数名でよい
クリック時のイベントなので、クリックされた位置の座標などを保持している



targetプロパティの利用

- イベントを呼び出した対象を参照する場合は、Eventオブジェクトのtargetプロパティを利用する

```
<button>佐藤</button>
<button>鈴木</button>
<button>山田</button>

<script>
// ボタンの名前を取得して、挨拶を表示する
const buttons = document.querySelectorAll("button");
buttons.forEach(button => {
  button.addEventListener("click", (event) => {
    const name = event.target.innerText;
    alert(`こんにちは、${name}さん`);
  });
});
</script>
```

querySelectorAll()はCSSセレクタを使い要素を選択するメソッドで、該当する要素をNodeオブジェクトの集合として取得できる

クリックされたボタンを指す

デフォルトの処理のキャンセル

- イベント内の処理で **preventDefault()** を使うと、そのイベントの**デフォルトの処理**をキャンセルすることができる
  - リンク先への遷移、フォームの送信など

記述例：送信前に確認を行う

```
<form name="registerForm" action="process.php" action="post">
  メールアドレス: <input type="email" name="email" required>
  <input type="submit" value="登録">
</form>
<script>
document.registerForm.addEventListener("submit", (event) => {
  if(!confirm("登録を確定してよろしいですか？")) {
    event.preventDefault(); // 送信をキャンセル
  }
});
</script>
```

confirmは確認用のウィンドウを表示させるメソッド
⇒ OKを選択するとtrueを返す

練習

- 練習04-1

Reactでのイベント処理

イベント処理の実装方法

- JSX内でReact要素に対し、イベントハンドラ属性を設定することで、イベント処理を実装する
  - querySelector()などによる要素選択やaddEventListener()によるイベントハンドラ設定は行わない
- HTMLのイベントハンドラ属性と似ているが、以下のような違いがある
  - 属性名はキャメルケースで記述する必要がある
    ⇒ onClick, onSubmit, onChange…等
  - 属性値は文字列ではなく、関数を記述する
  - 基本的には、通常のHTML/JavaScriptのイベントと同じ挙動になるが、一部異なるものも存在する
    ⇒ oninput属性を設定する場合、ReactではonChange属性にする

イベント処理の記述方法

方法１：無名関数を記述する

```
export default function Sample() {
  return <button onClick={() => alert("クリック")}>ボタン</button>;
}
```

方法２：定義済みの関数名を記述する

```
export default function Sample() {
  const handleClick = () => alert("クリック");

  return <button onClick={handleClick}>ボタン</button>;
}
```

練習

- 練習04-2

フォーム処理の実装方法

- フォームの入力値を取得する(扱う)には２通りのアプローチがある

1. useRefを使用する方法
   - フォームの送信ボタンが押下されるタイミングで、入力値を取得する場合に採用する
2. useStateを使用する方法
   - ユーザーの入力操作に対し、リアルタイムで処理を行いたい場合に採用する
     ⇒ ユーザーの入力に対し、リアルタイムでバリデーションを行い、メッセージを表示させたい場合など

必要に応じて、これらの方法を組み合わせて使用する

useRefフック

- useRefフックを使用することで、コンポーネント内のDOM要素を参照することができる
  - フォームから入力値を取得する際に利用されることが多い
- コンポーネント内で「状態」を保持する目的で利用することができる
  - currentプロパティを使用して値を更新する
  - useStateと違い、値の更新時に再レンダリングがされない

useRefとフォーム処理の実装

① useRef()の戻り値とフォームの部品を、ref属性を使い連携させる

例：氏名と年齢の入力値を取得する

```
const nameRef = useRef();
const ageRef = useRef();

return (<form onSubmit={handleSubmit}>
      <p>氏名: <input type="text" ref={nameRef} /></p>
      <p>年齢: <input type="number" ref={ageRef} /></p>
      <input type="submit" />
    </form>);
```

useRefとフォーム処理の実装

② currentプロパティを使い、フォーム部品のDOM要素を参照する

例: 氏名と年齢の入力値を取得する(前ページのコードに追記)

```
...
const handleSubmit = e => {
    e.preventDefault();
    const name = nameRef.current.value;
    const age = Number(ageRef.current.value);
    console.table({ name, age });
};

return (<form onSubmit={handleSubmit}>
      <p>氏名: <input type="text" ref={nameRef} /></p>
      <p>年齢: <input type="number" ref={ageRef} /></p>
...
```

氏名の入力欄のDOMを参照

年齢の入力欄のDOMを参照

初期値の設定

- 初期値はdefaultValue属性で設定する
  - value属性を設定してしまうと、入力値を変更することができない (onChange属性に対応するイベントハンドラが必要になる)

```
年齢：<input type="number" defaultValue={20} min={0} />

メールマガジン：
<input type="radio" name="magazine" defaultChecked /> 希望する
<input type="radio" name="magazine" /> 希望しない

お住まい：
<select defaultValue={1}>
    <option value={1}>東京都内</option>
    <option value={2}>東京都以外</option>
</select>

備考：<textarea cols="30" rows="10" defaultValue="特になし" />
```

ラジオボタンや
チェックボックスの場合、
defaultChecked

useStateとフォーム処理の実装

- value属性の値としてステート変数をセットする
  - これだけだと、ユーザーは何も入力できなくなってしまうので、onChange属性でステート値を更新する

例：氏名の入力値をステートとして保持

```
const [name, setName] = useState("");

return (<form>
      <p>氏名:
        <input type="text"
          value={name}
          onChange={e => setName(e.target.value)} />
      </p>
</form>);
```

value値にステート変数をセット

入力に変更がある度に、ステート値を更新する

select/option

- select/optionのフォーム部品に初期値を設定する場合は、select要素にvalue属性とonChange属性を設定する

「value="1" 国内」を初期値としてセット

例：エリア選択

```
const [area, setArea] = useState("1");

return (
  <select value={area} onChange={(e) => setArea(e.target.value)}>
    <option value="1">国内</option>
    <option value="2">海外</option>
  </select>
);
```

ラジオボタン

- ラジオボタンに初期値を設定する場合は、checked属性とonChange属性を設定する

例：エリア選択

「value="1" 国内」を初期値としてセット

```
const [area, setArea] = useState("1");

const handleChange = (e) => setArea(e.target.value);

return (<>
    <input type="radio" name="area" value="1"
           checked={area === "1"} onChange={handleChange} />国内
    <input type="radio" name="area" value="2"
           checked={area === "2"} onChange={handleChange} />海外
  </>);
```

checked属性は真偽値でセットする

チェックボックス

- 複数選択可能なチェックボックスの例

```
const [hobbies, setHobbies] = useState({
  cooking: false, sports: false, travel: false
});

const handleChange = (e) => {
   const { name, checked } = e.target;
   setHobbies(prev => ({ ...prev, [name]: checked }));
};

return (<>
    <input type="checkbox" name="cooking" value="1"
           checked={hobbies.cooking} onChange={handleChange} />料理
    <input type="checkbox" name="sports" value="2"
           checked={hobbies.sports} onChange={handleChange} />スポーツ
    <input type="checkbox" name="travel" value="3"
           checked={hobbies.travel} onChange={handleChange} />旅行
  </>);
```

ステートはオブジェクトとして管理

変化のあったチェックボックスから、name属性値とチェックの有無を取得

チェックボックスの状態を更新

練習

- 練習04-3
- 練習04-4