# Web講義5 ReactJS

基本的に<https://ja.react.dev/>を参照すること。

## Reactの機能

・コンポーネントの再利用

・インタラクティブな機能(ボタンなどの動作を指す)

課題1

公式サイトのチュートリアル「三目並べ」をやってみよう。

## コンポーネントの作成

・app/componentsフォルダを作成する。

Profileコンポーネントを作成してみよう。

・app/components/Profile.tsxファイルを作成する。

大文字で作成するのが流儀。

(基本形)

type Props={

}

export const Profile:FC<Props> = () => {

    return(<></>)

}

(Profile.tsx)

type Props={

    name:string;

    age:number;

    bio:string

}

export const Profile:FC<Props> = ({name,age,bio}) => {

    return(<div>

        <p>{name},{age}</p>

        <p>{bio}</p>

    </div>)

}

・コンポーネントの利用

(page.tsx)

export default function Page() {

  return (<main>

    <Profile/>

  </main>)

}

ただし、このままだとPropsが渡せていない。

export default function Page() {

  return (<main>

    <Profile name="灘校太郎" age={21} bio="よろ"/>

    <Profile name="灘校次郎" age={22} bio="よろ"/>

    <Profile name="灘校三郎" age={23} bio="よろ"/>

  </main>)

}

このようにコンポーネントを再利用できる。

課題2

Tailwindの知識を生かしてProfileコンポーネントのレイアウトを作成してみよう。思いつかない人は以下のレイアウトを再現してみよう。バブル チャート が含まれている画像

自動的に生成された説明

課題3

公式サイトの「UIの記述」を読み、実践しよう。

## リストのレンダリング(rendering)

JavaScriptではリストは配列やMapを指す。繰り返し処理をして、Profileの表示を簡略化してみよう。

const people = [

  {

    name: "灘校太郎",

    age: 20,

    bio: "僕は灘校です。"

  },

  {

    name: "灘校次郎",

    age: 18,

    bio: "僕は次郎です よろしくね"

  },

  {

    name: "灘校三郎",

    age: 16,

    bio: "僕は三郎です。"

  }

]

export default function Page() {

  return (<main>

    {

      people.map((person, index) =>

        <Profile key={index}

          name={person.name}

          age={person.age}

          bio={person.bio} />)

    }

  </main>)

}

課題5

フルーツの配列「[“リンゴ”,”オレンジ”…]」を作成し、それらについて「(フルーツ)大好き」というpタグの文章を生成してみよう。

## Reactコンポーネントの変数

次のようなコンポーネントを考えてみる。

(components/Counter.tsx)

type Props={}

export const Counter:FC<Props>=()=>{

    const [num,setNum]=useState(0)

    return(<div>

        <p>Counter</p>

        <p>{num}</p>

        <button onClick={()=>setNum(num+1)}>+</button>

    </div>)

}

export const Counter:FC<Props>=()=>{

    let num=0

    return(<div>

        <p>Counter</p>

        <p>{num}</p>

        <button onClick={()=>{

            num+=1

        }}>+</button>

    </div>)

}

課題6

二つの場合で実際に動作を確認しよう。

Point: JSX内で使用する変数は状態管理をする必要がある。

Counterコンポーネントを次のように書き換えてみよう。

export const Counter:FC<Props>=()=>{

    const [num,setNum]=useState(0)

    useEffect(()=>{

        document.title=`Counter: ${num}`

    },[num])

    return(<div>

        <p>Counter</p>

        <p>{num}</p>

        <button onClick={()=>setNum(num+1)}>+</button>

    </div>)

}

課題7

useEffectについて調べてみよう。

発展

export default function Page() {

  return (<main>

    <Timer/>

  </main>)

}

const Timer:FC=()=>{

  const now=new Date()

  return(

    <p>{now.toString()}</p>

  )

}

Testコンポーネントを分析してみるとnow変数を宣言し、現在の時間をpタグとして返している。

課題8

ブラウザで動作を確認しよう。再更新のショートカットキーはCtrl+Rである。

このように、Testコンポーネントは最初の描画時しか更新されない。更新するためには値が変わったことをReactに通知する必要がある。

実際に毎秒更新するのは少し難しいためコードサンプルだけ置いておく。

export const Timer: FC = () => {

  const [now, setNow] = useState(new Date())

  const interval = useRef<NodeJS.Timeout>()

  useEffect(() => {

    interval.current = setInterval(() => {

      setNow(new Date())

    }, 1000)

    return () => {

      clearInterval(interval.current)

    }

  }, [])

  return (

    <p>{now.toString()}</p>

  )

}

課題9

useRefフックについて調べてみよう。

このようにフックを使うことでたくさんの処理ができる。また、フックを自作していくこともできるので、興味のある人はやってみよう。

例えば、chakra-uiというライブラリの中にuseToastというフックがある。

公式サイトを見てみよう。