React

학습 목표

React 개발 환경, React 속성 등 하나씩 배워가면서

간단한 웹 어플리케이션을 만들어 본다.

개발 예제

To do list

할 일을 입력해주세요 등록

이것도 해야 되고 삭제

저것도 해야 되고 삭제

그것도 해야 되고 삭제

언제 다하나 삭제

목차

1. 왜 React인가?

기존 자바스크립트

- DOM을 가져오고 갱신하는 작업은 느림
- 상태가 바뀔 때마다 페이지 전체를 렌더링하는 애플리케이션의 경우 성능이 떨어짐

리액트

- 페이스북과 인스타그램에 적용하기 위해 만들어짐
- 가상 DOM을 이용해서 페이지에 렌더링 된 전체 DOM을 읽음
- state와 props 값에 변화가 있으면 해당되는 부분만 렌더링
- 리플로우나 불필요한 DOM조작을 최소화 하기 때문에 성능 향상
- 가상 돔의 diff 체크가 리액트에서 제공되는 핵심 기능

리액트와 함께 사용되는 것

- 웹 프레임워크를 MVC로 구분한다고 했을 때 리액트는 V(View)에 해당
- 개발 파일이 많아짐에 따라 M(Model)에 해당하는 react-redux와
- C(Controller)에 해당하는 react-router를 필요에 따라 추가로 사용

2. 개발환경 구축

Git 설치

- https://git-scm.com/downloads

NodeJs 설치

- https://nodejs.org/ko/download/
- lts 버전으로 설치

```
→  todo_react git:(master) git --version
git version 2.6.2
→  todo_react git:(master) node -v
v6.4.0
→  todo_react git:(master) npm -v
3.10.3
```

저장소 clone

- git clone https://github.com/apple77y/todo_react.git
- template.html 확인

2. 개발환경 구축

브랜치 변경

- git checkout -b origin/00-devSetting

dependencies

- react: 리액트 컴포넌트를 만들어 주는 모듈
- react-dom: 만들어진 컴포넌트를 실제 DOM에 붙여주는 모듈

devDependencies

- babel-core: 트랜스파일을 해주는 모듈의 코어
- babel-loader: 웹팩에서 loader로 babel 불러오기 위한 모듈
- babel-preset-react: JSX로 개발된 파일을 일반 JS로 변환하기 위한 모듈
- react-hot-loader: 웹팩 개발 서버에서 HMR을 해줄 모듈
- webpack: require(모듈화)를 사용하기 위해 쓰는 모듈
- webpack-dev-server: 개발 서버를 사용하기 위해 쓰는 모듈

실습

HTML 파일을 생성

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Todo</title>
</head>
<body>
    <div id="app"></div>
    <script src="./dist/bundle.js"></script>
</body>
</html>
```

- 요소에 리액트 컴포넌트가 그려질 예정
- bundle.js 파일이 웹팩으로 트랜스파일 된 빌드 산출물

3. Hello World

Hello World를 HTML 파일에 출력

```
import React from ‘react';
import ReactDOM from 'react-dom';

if (module.hot) {
 module.hot.accept();
}

ReactDOM.render(
 <div>Hello World</div>, document.getElementById('app')
);
```

- ReactDOM은 React에 의존성을 가짐
- module.hot.accept는 개발용 HMR(Hot Module Replacement)를 위한 코드

4. JSX

JSX = JavaScript + XML의 약자

- HTML처럼 보이게 하는 효과
- 사용하지 않아도 되지만, 실제로는 사용할 수 밖에 없음

주의 사항

- XML이라서 HTML처럼 루트가 꼭 있어야 됨
- 중괄호 안에서는 자바스크립트를 쓸 수 있음
- 열림 태그와 닫힘 태그가 반드시 있음
  - ex) <hr />, <input />,
- 중괄호 안에서 조건문 if는 쓸 수 없음
  - **로직 분리** 혹은 **삼항 연산자** 사용을 추천
- 자바스크립트의 예약어와 겹치는 예약어는 다른 걸 써야됨
  - div class -> className
  - label for -> htmlFor

실습

template.html의 내용을 jsx로 변환

- index.html의 head에 css 파일 추가 필요

```jsx
import React from 'react';
import ReactDOM from 'react-dom';

if (module.hot) {
    module.hot.accept();
}

ReactDOM.render(
    <div className="container">
        <div className="page-header">
            <h1>To do list</h1>
        </div>
        <div className="input-group input-group-lg">
            <input type="text" className="form-control" placeholder="할 일을 입력해주세요" />
                <span className="input-group-btn">
                    <button className="btn btn-primary" type="button">등록</button>
                </span>
        </div>
        <hr/>
            <ul>
                <li>
                    <span>이것도 해야 되고</span>
                    <span className="btn-container"><a href="#">삭제</a></span>
                </li>
                <li>
                    <span>저것도 해야 되고</span>
                    <span className="btn-container"><a href="#">삭제</a></span>
                </li>
                <li>
                    <span>그것도 해야 되고</span>
                    <span className="btn-container"><a href="#">삭제</a></span>
                </li>
                <li>
                    <span>언제 다 하나</span>
                    <span className="btn-container"><a href="#">삭제</a></span>
                </li>
            </ul>
    </div>,
    document.getElementById('app')
);
```

실습

상수 값은 변수로 추출

```
import React from 'react';
import ReactDOM from 'react-dom';

if (module.hot) {
    module.hot.accept();
}

const text = 'To do list';
const todo = [
    '이것도 해야 되고',
    '저것도 해야 되고',
    '그것도 해야 되고',
    '언제 다 하나'
];

ReactDOM.render(
    <div className="container">
        <div className="page-header">
            <h1>{text}</h1>
        </div>
        <div className="input-group input-group-lg">
            <input type="text" className="form-control" placeholder="할 일을 입력해주세요" />
                <span className="input-group-btn">
                    <button className="btn btn-primary" type="button">등록</button>
                </span>
        </div>
        <hr/>
            <ul>
                <li>
                    <span>{todo[0]}</span>
                    <span className="btn-container"><a href="#">삭제</a></span>
                </li>
                <li>
                    <span>{todo[1]}</span>
                    <span className="btn-container"><a href="#">삭제</a></span>
                </li>
                <li>
                    <span>{todo[2]}</span>
                    <span className="btn-container"><a href="#">삭제</a></span>
                </li>
                <li>
                    <span>{todo[3]}</span>
                    <span className="btn-container"><a href="#">삭제</a></span>
                </li>
            </ul>
    </div>,
    document.getElementById('app')
);
```

실습

view의 관심사 별로 분리

```
import React from 'react';
import ReactDOM from 'react-dom';

if (module.hot) {
    module.hot.accept();
}

const text = 'To do list';

const Title = (
    <div className="page-header">
        <h1>{text}</h1>
    </div>
);

const AddLi = (
    <div className="input-group input-group-lg">
        <input type="text" className="form-control" placeholder="할 일을 입력해주세요" />
                <span className="input-group-btn">
                    <button className="btn btn-primary" type="button">등록</button>
                </span>
    </div>
);

const todos = [
    '이것도 해야 되고',
    '저것도 해야 되고',
    '그것도 해야 되고',
    '언제 다 하나'
];

const TodoLi = todos.map((todo, i) => {
    return (
    <li key={'todo' + i}>
        <span>{todo}</span>
        <span className="btn-container"><a href="#">삭제</a></span>
    </li>

    );
});


ReactDOM.render(
    <div className="container">
        {Title}
        {AddLi}
        <hr/>
            <ul>
                {TodoLi}
            </ul>
    </div>,
    document.getElementById('app')
);
```

5. React Component

class … extends React.Component 로 생성

- 반환 값은 리액트 객체
  - 재사용 가능
- render 메소드를 통해 JSX를 반환
- props와 state 사용 가능
- 제공되는 메소드
  - ES5: getInitialState -> ES6: 생성자에서 선언
  - ES5: getDefaultProps -> ES6: 클래스 객체에 속성 추가
  - componentWillMount
  - componentDidMount
  - componentWillReceiveProps
  - componentWillUpdate
  - componentDidUpdate
  - componentWillUnmount
  - shouldComponentUpdate
  - render 메소드는 필수 항목

실습

AddLi.js

```
import React, {Component} from 'react';

class AddLi extends Component {
    render() {
        return (
            <div className="input-group input-group-lg">
                <input type="text" className="form-control" placeholder="할 일을
입력해주세요" />
                <span className="input-group-btn">
                    <button className="btn btn-primary" type="button">등록</
button>
                </span>
            </div>
        );
    }
}

export default AddLi;
```

- export default를 해야 외부 파일에서 require로 접근 가능

실습

Title.js

```
import React, {Component} from 'react';

const text = 'To do list';

class Title extends Component {
    render() {
        return (
            <div className="page-header">
                <h1>{text}</h1>
            </div>
        );
    }
}

export default Title;
```

실습

TodoLi.js

```
import React, {Component} from 'react';

class TodoLi extends Component {
    render() {
        const TodoLi = todos.map((todo, i) => {
            return (
                <li key={'todo' + i}>
                    <span>{todo}</span>
                    <span className="btn-container"><a href="#">삭제</a></span>
                </li>

            );
        });

        return (
            <div>
                {TodoLi}
            </div>
        );
    }
}

export default TodoLi;
```

실습

App.js

```
import Title from './component/Title';
import AddLi from './component/AddLi';
import TodoLi from './component/TodoLi';

if (module.hot) {
    module.hot.accept();
}

ReactDOM.render(
    <div className="container">
        <Title/>
        <AddLi/>
        <hr/>
            <ul>
                <TodoLi/>
            </ul>
    </div>,
    document.getElementById('app')
);
```

- import라는 예약어로 export 된 모듈에 접근

실습

container/Todo.js

```
class Todo extends Component {
    render() {
        return (
            <div className="container">
                <Title/>
                <AddLi/>
                <hr/>
                <ul>
                    <TodoLi/>
                </ul>
            </div>
        );
    }
}

export default Todo;
```

- 역할에 따라 Container와 Component를 구분

6. Props와 State

Props

- 부모 컴포넌트로부터 받은 데이터
- 쓰기는 불가능. 읽기만 가능
- React.PropTypes를 통해 데이터 형(type) 체크 가능

State

- 컴포넌트 안에서 변경이 가능한 데이터
- 쓰기 가능
  - state는 immutable하기 때문에 setState 메소드를 통해 갱신
- 컴포넌트는 state를 최소한으로 사용
- state를 변경하는 로직은 Container에서 최대한 담당

실습

Todo.js: 각 컴포넌트로 데이터를 전달

```
const text = 'To do list';

const todos = [
    '이것도 해야 되고',
    '저것도 해야 되고',
    '그것도 해야 되고',
    '언제 다 하나'
];

class Todo extends Component {
    render() {
        return (
            <div className="container">
                <Title text={text} />
                <AddLi/>
                <hr/>
                <ul>
                    <TodoLi todos={todos}/>
                </ul>
            </div>
        );
    }
}
```

- 전달 받을 props 이름 = {전달할 데이터}

실습

Title.js: 부모로부터 전달받은 데이터(this.props.text)를 출력

```
class Title extends Component {
    render() {
        return (
            <div className="page-header">
                <h1>{this.props.text}</h1>
            </div>
        );
    }
}
```

실습

TodoLi.js: 부모로부터 전달받은 데이터(this.props.todos)를 출력

```
class TodoLi extends Component {
    render() {
        const TodoLi = this.props.todos.map((todo, i) => {
            return (
                <li key={'todo' + i}>
                    <span>{todo}</span>
                    <span className="btn-container"><a href="#">삭제</a></span>
                </li>

            );
        });

        return (
            <div>
                {TodoLi}
            </div>
        );
    }
}
```

실습

Todo.js: 초기 상수 값을 생성자에서 관리

```
class Todo extends Component {
    constructor() {
        this.state = {
            text: 'To do list',
            todos: [
                '이것도 해야 되고',
                '저것도 해야 되고',
                '그것도 해야 되고',
                '언제 다 하나'
            ]
        };
    }

    render() {
        return (
            <div className="container">
                <Title text={this.state.text} />
                <AddLi/>
                <hr/>
                <ul>
                    <TodoLi todos={this.state.todos}/>
                </ul>
            </div>
        );
    }
}
```

- ES6는 생성자 함수에서 초기 state 값 설정

실습

Todo.js: todoLi 반복문 로직을 컴포넌트가 아닌 컨테이너에서 수행

```
const todoLi = this.state.todos.map((todo, i) => {
    return <TodoLi todo={todo} key={'todo' + i} />;
});

return (
    <div className="container">
        <Title text={this.state.text} />
        <AddLi/>
        <hr/>
        <ul>
            {todoLi}
        </ul>
    </div>
);
```

7. React Component의 Lifecycle

컴포넌트의 생명주기

- 리액트 컴포넌트가 DOM에 그려지는 과정을 분리
- 그 과정(= 시점)들을 생명주기라 부름
- props와 state 값이 바뀔 때마다 필요한 부분만 그려짐
- 생명주기 메소드는 총 7개
- https://facebook.github.io/react/docs/component-specs-ko-KR.html#생명주기-메소드

생명주기 메소드의 활용 예

- componentDidMount에서의 ajax 호출
- componentWillUnmount에서의 이벤트 해제

실습

Todo.js: 컴포넌트 생명주기를 확인

```
componentWillMount() {
    console.log('componentWillMount');
}

componentDidMount() {
    console.log('componentDidMount');
}

componentWillReceiveProps() {
    console.log('componentWillReceiveProps');
}

componentWillUpdate() {
    console.log('componentWillUpdate');
}

componentDidUpdate() {
    console.log('componentDidUpdate');
}

componentWillUnmount() {
    console.log(‘componentWillUnmount');
}
```

실습

Todo.js: 초기 값을 서버에서 받아온다고 가정

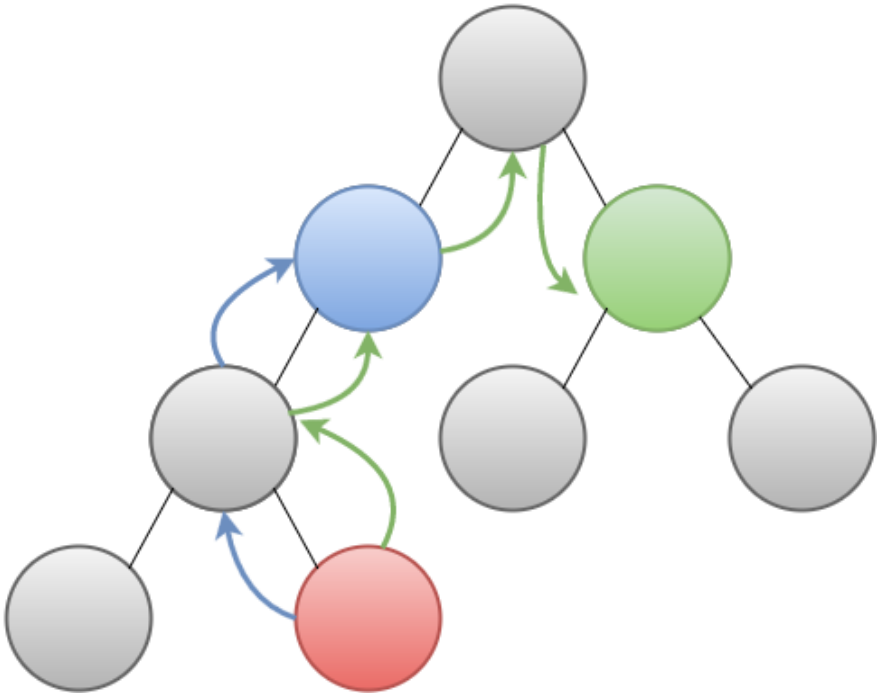
```
constructor(props) {
    super(props);

    this.state = {
        text: '',
        todos: []
    };
}

componentDidMount() {
    const text = 'To do list';
    const todos = [
        '이것도 해야 되고',
        '저것도 해야 되고',
        '그것도 해야 되고',
        '언제 다 하나'
    ];

    this.setState((prevState) => {
        return {
            text,
            todos
        };
    });
}
```

- text, todos를 ajax로 받아왔다고 가정

8. Event

이벤트 처리

- DOM 이벤트 리스너는 카멜 케이스로
- DOM을 다룰 때는 ref(reference) 속성을 이용해서 가상 DOM을 참조
  - native 혹은 jQuery로 DOM을 다룰 수 있지만 지양
  - https://facebook.github.io/react/docs/more-about-refs-ko-KR.html
- 상위 컴포넌트로 데이터를 전달할 때에는 콜백 함수를 이용



실습

AddLi.js: 등록 버튼을 눌렀을 때의 이벤트

```
onClickAddButton() {
    this.props.handleAddedData(this.inputBox.value);
    this.inputBox.value = '';
    this.inputBox.focus();
}

render() {
    return (
        <div className="input-group input-group-lg">
            <input type="text" className="form-control" placeholder="할 일을
입력해주세요"
                   ref={input => {this.inputBox = input;}}/>
            <span className="input-group-btn">
                <button className="btn btn-primary" type="button"
onClick={this.onClickAddButton}>등록</button>
            </span>
        </div>
    );
}
```

실습

Todo.js: 컨테이너의 콜백함수에서 데이터 수신

```
constructor() {
     this.handleAddedData = this.handleAddedData.bind(this);
}

handleAddedData(text) {
    this.setState((prevState) => {
        const {todos} = prevState;
        todos.push(text);

        return {todos};
    });
}

render: function () {
     return (
         <div className="container">
             <Title text={this.state.text}/>
             <AddLi handleAddedData={this.handleAddedData} />
             <hr/>
             <ul>
                 {todoLi}
             </ul>
         </div>
     );
}
```

- state의 데이터는 immutable하기 때문에 새로운 객체를 참조

실습

TodoLi.js: 삭제 버튼을 눌렀을 때의 이벤트

```
class TodoLi extends Component {
    constructor(props) {
        super(props);

        this.onClickRemoveButton = this.onClickRemoveButton.bind(this);
    }

    onClickRemoveButton() {
        this.props.handleRemovedData(this.props.todo);
    }

    render() {
        return (
            <li>
                <span>{this.props.todo}</span>
                <span className="btn-container">
                    <a href="#" onClick={this.onClickRemoveButton}>삭제</a>
                </span>
            </li>
        );
    }
}
```

실습

Todo.js: 컨테이너의 콜백함수에서 데이터 수신

```
constructor() {
    this.handleRemovedData = this.handleRemovedData.bind(this);
}

handleRemovedData(text) {
    this.setState((prevState) => {
        const {todos} = prevState;
        const index = todos.indexOf(text);
        todos.splice(index, 1);

        return {todos};
    })
}

render: function () {
    var todoLi = this.state.todos.map(function (todo, i) {
        return <TodoLi todo={todo} key={'todo' + i}
                  handleRemovedData={this.handleRemovedData}/>;
    });
}
```

- state의 데이터는 immutable하기 때문에 새로운 객체를 참조

9. Stateless Component

상태 값이 없는 컴포넌트

- 페이스북에서 지향하는 컴포넌트 방향
- 재활용하기 쉬움
- 리액트 코어 내에서도 빠른 경로 = 성능 이점
- 단, 컴포넌트 내에서 this.state, 생성주기 함수, ref 사용이 없어야 됨
- https://facebook.github.io/react/docs/reusable-components-ko-KR.html#상태를-가지지-않는-함수

실습

Title.js: 순수 함수 형태로 리팩토링

```
import React from 'react';

const Title = ({text}) => (
    <div className="page-header">
        <h1>{text}</h1>
    </div>
);

export default Title;
```

실습

TodoLi.js: 순수 함수 형태로 리팩토링

```
import React from 'react';

const TodoLi = ({handleRemovedData, todo}) => {
    const onClickRemoveButton = () => {
        handleRemovedData(todo);
    };

    return (
        <li>
            <span>{todo}</span>
                <span className="btn-container">
                    <a href="#" onClick={onClickRemoveButton}>삭제</a>
                </span>
        </li>
    );
};

export default TodoLi;
```

10. PropTypes

Prop 검증

- 부모로부터 받은 데이터의 형(type)이 올바른지 체크
- 개발 모드에서만 검사 수행됨

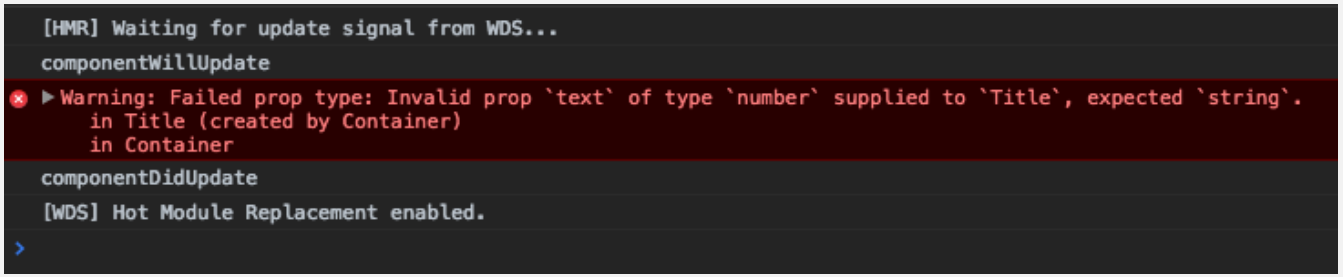
검증 가능한 형(type) 사용 예

- 배열: PropTypes.array
- 불리언: PropTypes.bool
- 함수: PropTypes.func
- 숫자: PropTypes.number
- 객체: PropTypes.object
- 문자: PropTypes.string
- 필수 값은 isRequired를 붙임

실습

Title.js: 객체의 속성 값으로 propTypes을 주입

```
import PropTypes from 'prop-types';

const Title = ({text}) => (
     (……생략)
);

Title.propTypes = {
    text: PropTypes.string
};
```

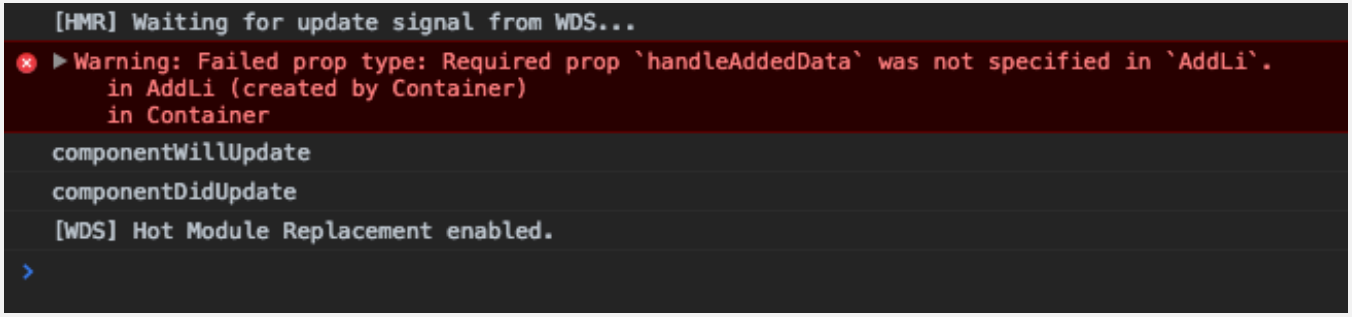


- text의 값으로 string이 아닌 number가 넘어왔을 경우

실습

AddLi.js: 객체의 속성 값으로 propTypes을 주입

```
import PropTypes from 'prop-types';

class AddLi extends Component {
    (……생략)
}

AddLi.propTypes = {
    handleAddedData: PropTypes.func.isRequired
};
```

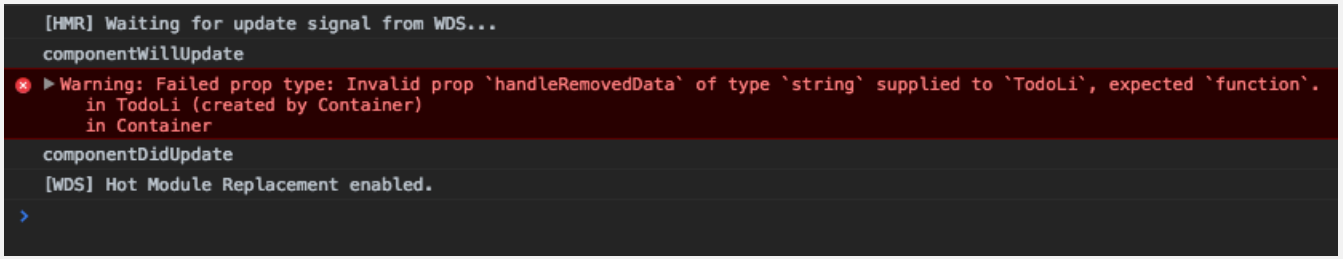


- 필수 값이 안 넘어왔을 경우

실습

TodoLi: 객체의 속성 값으로 propTypes을 주입

```
import PropTypes from 'prop-types';

const TodoLi = ({handleRemovedData, todo}) => {
    (……생략)
};

TodoLi.propTypes = {
    handleRemovedData: React.PropTypes.func.isRequired,
    todo: React.PropTypes.string.isRequired
};
```



- handleRemovedData의 값으로 func 대신 string이 넘어왔을 경우