React tips

while building
facebook-scale application


Reese's

ShihChi Huang

Container Component

Container Component

Flux ReduceStore

Container Component

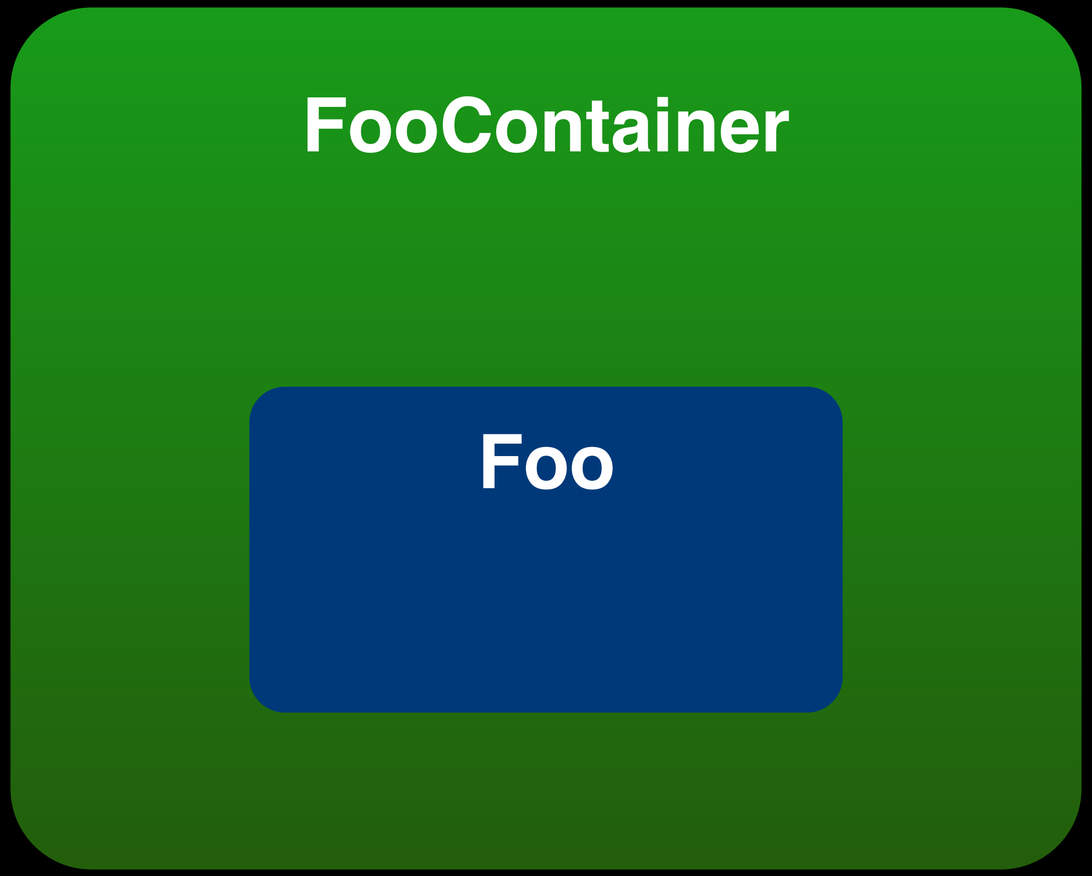
Flux ReduceStore

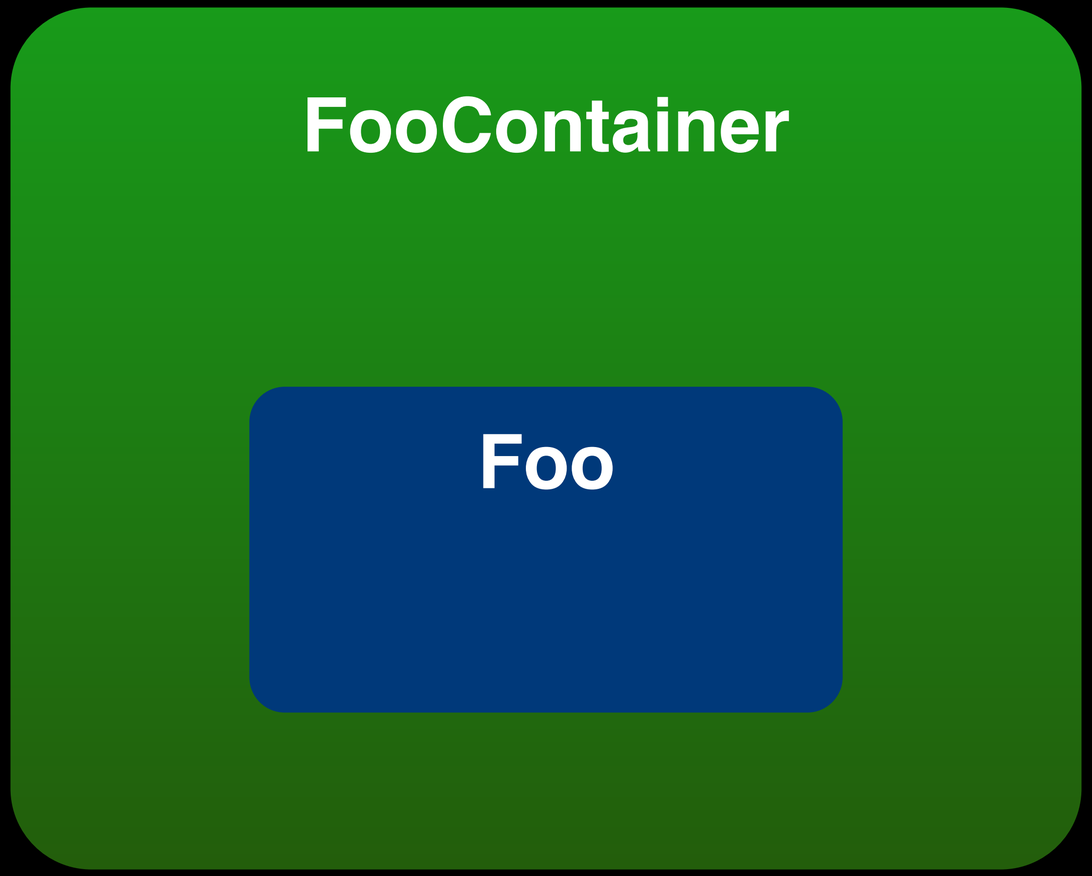
Functional *

what

is Container Component

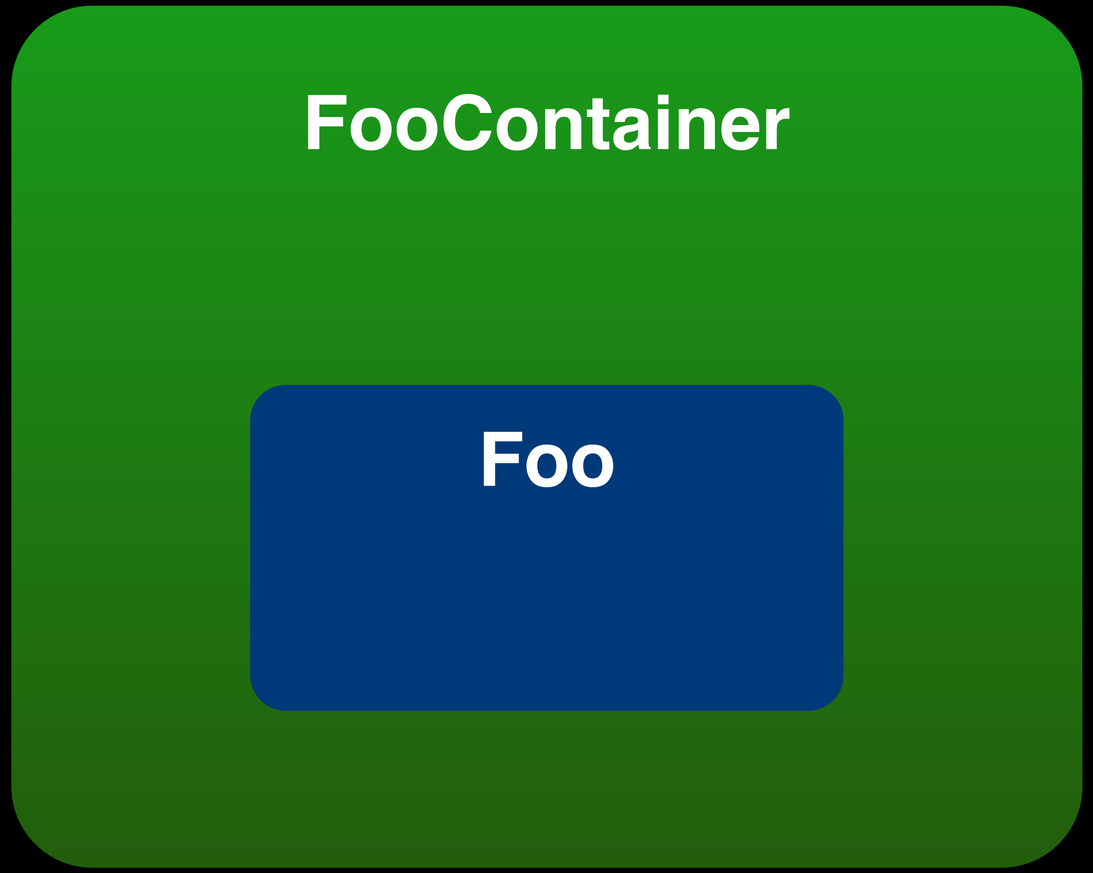

FooContainer


FooContainer
Foo


FooContainer
Foo

handle data
loading / error

render



how

to create
Container Component

```
class FooContainer extends Component {
  constructor(props) {
    super(props);
    this.state = { data: [] };
  }
  render() {
    return (
      <Foo {...this.state} />
    );
  }
}
```

```
class FooContainer extends Component {
  static getStores() {
    return [FooStore];
  }
  static calculateState() {
    return { data: FooStore.getData() };
  }
  render() {
    return (
      <Foo {...this.state} />
    );
  }
}
```

why

Container Component

Container Component

Container Component

- manage data/state

Container Component

- manage data/state
- UI logic

Container Component

- manage data/state
- UI logic
- reusable

Container Component

- manage data/state
- UI logic
- reusable
- needed tests

UI Component

- manage data/state
- UI logic
- reusable
- needed tests

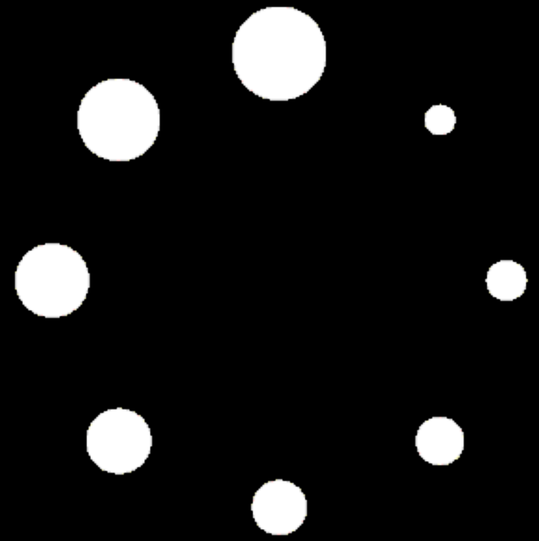
```
class TodoApp extends Component {
  constructor(props) {
    super(props);
    this.state = { todos: [] };
  }
  componentDidMount() {
    fetch('/todos.json').then(todos => {
      this.setState({ todos });
    });
  }
  render() {
    const todos = this.state.todos
      .map(todo => <Todo {...todo} />);
    return <ul>{todos}</ul>;
  }
}
```

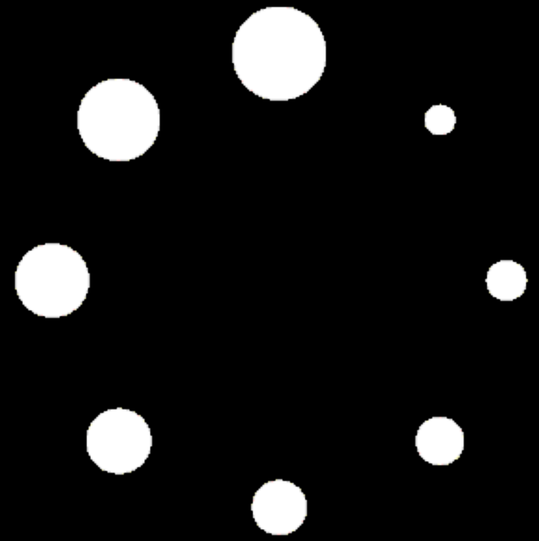
```
class TodoApp extends Component {
  constructor(props) {
    super(props);
    this.state = { todos: [] };
  }
  componentDidMount() {
    fetch('/todos.json').then(todos => {
      this.setState({ todos });
    });
  }
  render() {
    const todos = this.state.todos
      .map(todo => <Todo {...todo} />);
    return <ul>{todos}</ul>;
  }
}
```

```
class TodoApp extends Component {
  componentDidMount() {
    fetch('/todos.json').then(




    );
  }
}
```

```
class TodoApp extends Component {
  componentDidMount() {
    fetch('/todos.json').then(
      todos => {
        this.setState({
          error: null,
          todos
        });
      },



    );
  }
}
```

```
class TodoApp extends Component {
  componentDidMount() {
    fetch('/todos.json').then(
      todos => {
        this.setState({
          error: null,
          todos
        });
      },
      error => {
        this.setState({ error });
      }
    );
  }
}
```





```
class TodoApp extends Component {
  componentDidMount() {


  }
```

```
class TodoApp extends Component {
  componentDidMount() {
    this.setState({ isLoaded: false });
  }
}
```

```
class TodoApp extends Component {
  componentDidMount() {
    this.setState({ isLoaded: false });
    fetch('/todos.json').then(
      todos => {
        this.setState({
          error: null,
          todos,
          isLoaded: true,
        });
      },
      error => {
        this.setState({ error });
      }
    );
  }
}
```

timeout/retry

timeout/retry
batch

timeout/retry
batch
filter by *

```
class TodoContainer extends Component {
  componentDidMount() {
    this.setState({ isLoading: true });
    fetch('/todos.json').then(
      todos => {
        this.setState({
          error: null,
          todos,
          isLoading: false
        });
      },
      error => this.setState({ error })
    );
  }
  render() {

  }
}
```

```
class TodoContainer extends Component {
  componentDidMount() {
    this.setState({ isLoading: true });
    fetch('/todos.json').then(
      todos => {
        this.setState({
          error: null,
          todos,
          isLoading: false
        });
      },
      error => this.setState({ error })
    );
  }
  render() {
    return <Todo {...this.state} />;
  }
}
```

```
class Todo extends Component {
  render() {




  }
}
```

```
class Todo extends Component {
  render() {
    const { error, isLoading, todos }
      = this.props;

  }
}
```

```
class Todo extends Component {
  render() {
    const { error, isLoading, todos }
      = this.props;
    if (error) { return <ErrorPage />; }









  }
}
```

```
class Todo extends Component {
  render() {
    const { error, isLoading, todos }
      = this.props;
    if (error) { return <ErrorPage />; }
    if (isLoading) { return <Spinner />; }









  }
}
```

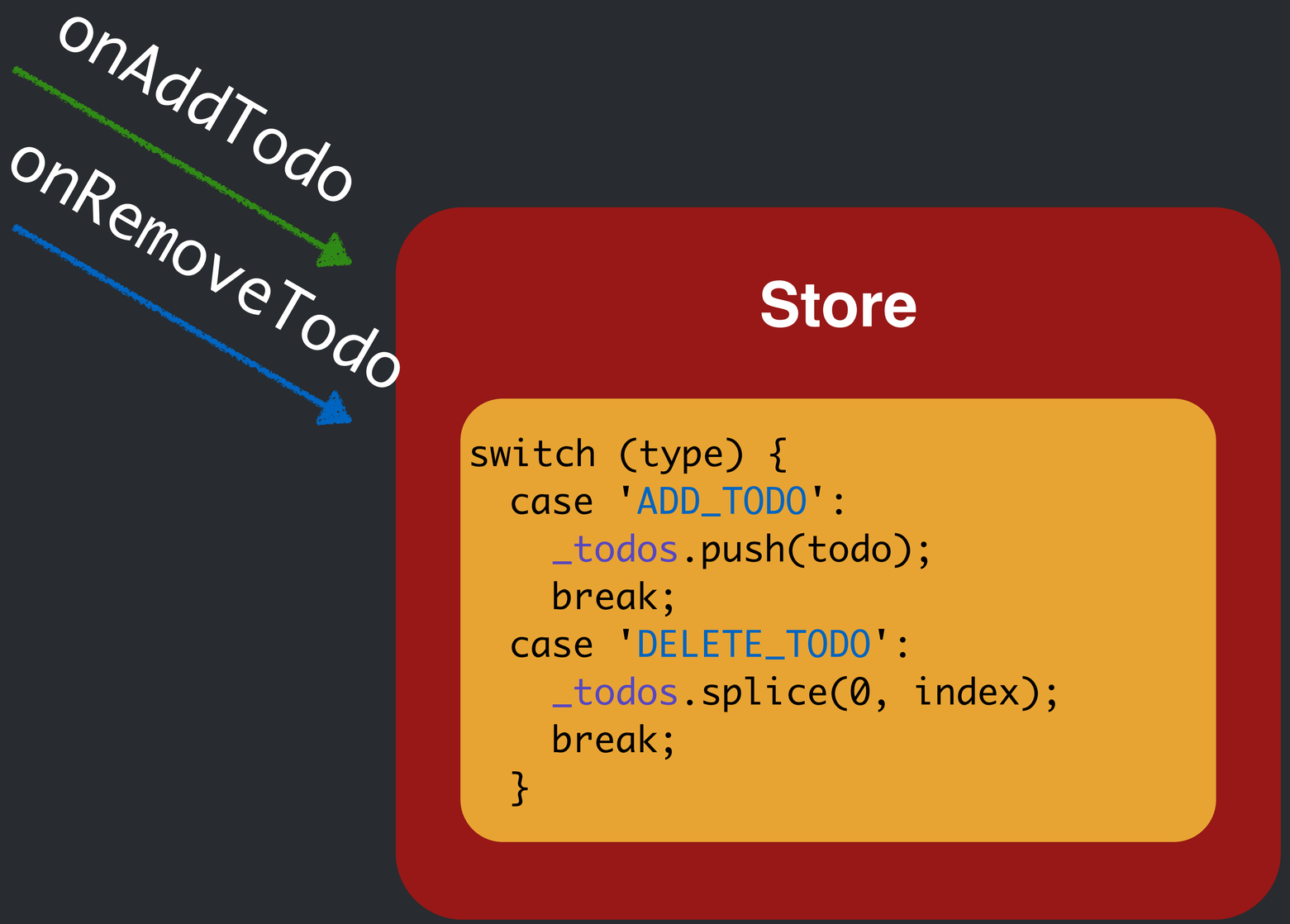
```
class Todo extends Component {
  render() {
    const { error, isLoading, todos }
      = this.props;
    if (error) { return <ErrorPage />; }
    if (isLoading) { return <Spinner />; }
    if (!todos.length) {
      return <EmptyResult />;
    }




  }
}
```

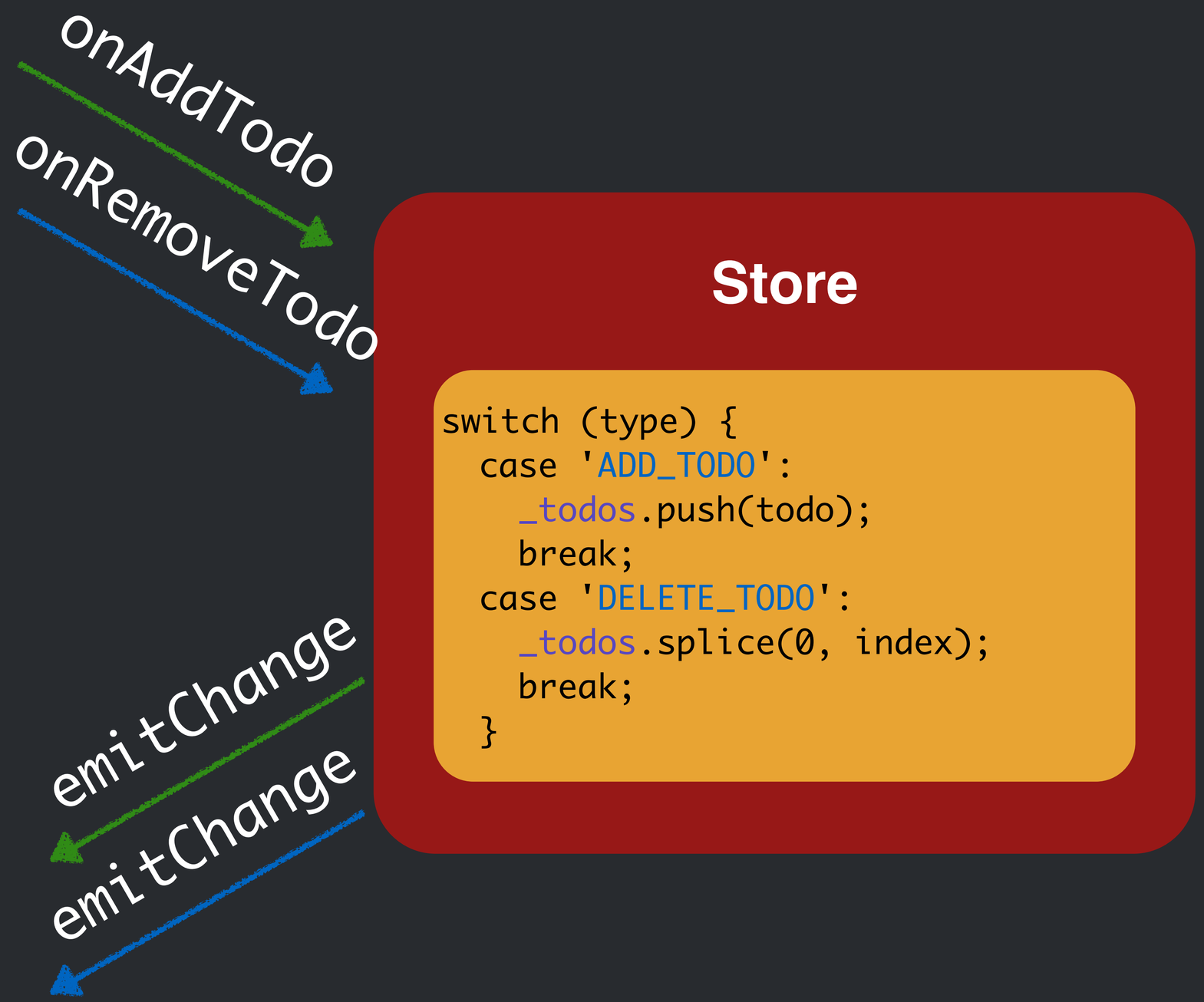
```
class Todo extends Component {
  render() {
    const { error, isLoading, todos }
      = this.props;
    if (error) { return <ErrorPage />; }
    if (isLoading) { return <Spinner />; }
    if (!todos.length) {
      return <EmptyResult />;
    }
    const lists = todos.map(todo => {
      return <TodoItem {...todo} />;
    });
    return <ul>{lists}</ul>;
  }
}
```


FLUX STORE
REMEMBER?
memegenerator.net

Store

```
switch (type) {
  case 'ADD_TODO':
    _todos.push(todo);
    break;
  case 'DELETE_TODO':
    _todos.splice(0, index);
    break;
  }
```


onAddTodo
onRemoveTodo
Store
switch (type) {
case 'ADD_TODO':
_todos.push(todo);
break;
case 'DELETE_TODO':
_todos.splice(0, index);
break;
}


onAddTodo
onRemoveTodo
Store
switch (type) {
case 'ADD_TODO':
_todos.push(todo);
break;
case 'DELETE_TODO':
_todos.splice(0, index);
break;
}
emitChange
emitChange

```
import {EventEmitter} from 'events';
const TodoStore = Object.assign(
  {}, EventEmitter.propotype
);
```

```
import {EventEmitter} from 'events';
const TodoStore = Object.assign(
  {}, EventEmitter.propotype
);
TodoStore.dispatchToken =
  MyDispatcher.register(action => {




    }
  });
```

```
import {EventEmitter} from 'events';
const TodoStore = Object.assign(
  {}, EventEmitter.propotype
);
TodoStore.dispatchToken =
  MyDispatcher.register(action => {
    switch (action.type) {
    case 'ADD_TODO':
      this._todos.push(action.todo);
      TodoStore.emitChange();
      break;




    }
  });
```

```
import {EventEmitter} from 'events';
const TodoStore = Object.assign(
  {}, EventEmitter.propotype
);
TodoStore.dispatchToken =
  MyDispatcher.register(action => {
    switch (action.type) {
    case 'ADD_TODO':
      this._todos.push(action.todo);
      TodoStore.emitChange();
      break;
    case 'DELETE_TODO':
      this._todos.splice(0, action.index);
      TodoStore.emitChange();
      break;
    }
  });
```

Redux: Atomic Flux with Hot Reloading (github.com)

186 points by monort 238 days ago | past | web | 44 comments

staltz 238 days ago

Redux is a major improvement over Flux libraries because it removes a lot of boilerplate while not removing functionality (in fact, enabling more features). In Flux libraries, it's common for an entity to take inputs and simply produce outputs. That's not the use case for a class. It's the use case for a function, and that's why Redux removes boilerplate.

However, I think it's time we stop calling everything Flux. Just because an architecture is unidirectional doesn't mean it's Flux. Facebook described it clearly as an architecture structured with: Dispatcher, Stores, Actions, Action Creators, and Components (sometimes even with the distinction of View and Controller View). Redux is Flux-inspired, but has significant differences. Maybe we should call it just Flux-inspired architecture. The distinction is important because there are other unidirectional data flow architectures such as in Elm (https://github.com/evancz/...) and Cycle.js (http://cycle.js.org). We might be creating confusion when mentioning "Flux" and meaning different things.

Hacker News
new | comments | show | ask | jobs | submit
login

Redux: Atomic Flux with Hot Reloading (github.com)

186 points by monort 238 days ago | past | web | 44 comments

staltz 238 days ago

Redux is a major improvement over Flux libraries because it removes a lot

In Flux libraries, it's common for an entity to take inputs and simply produce outputs. That's not the use case for a class. It's the use case for a function

and that's why Redux removes

However, I think it's time we stop calling everything Flux. Just because an architecture is unidirectional doesn't mean it's Flux. Facebook described it clearly as an architecture structured with: Dispatcher, Stores, Actions, Action Creators, and Components (sometimes even with the distinction of View and Controller View). Redux is Flux-inspired, but has significant differences. Maybe we should call it just Flux-inspired architecture. The distinction is important because there are other unidirectional data flow architectures such as in Elm (https://github.com/evancz/...) and Cycle.js (http://cycle.js.org). We might be creating confusion when mentioning "Flux" and meaning different things.

Hacker News
new | comments | show | ask | jobs | submit
login

Redux: Atomic Flux with Hot Reloading (github.com)

186 points by monort 238 days ago | past | web | 44 comments

staltz 238 days ago

Redux is a major improvement over Flux libraries because it removes a l

In Flux libraries, it's common for an entity to take inputs and simply produce outputs. That's not the use case for a class. It's the use case for a function

and that's why Redux removes

However, I think it's time we stop calling everything Flux. Just because an architecture is unidirectional doesn't mean it's Flux. Facebook described it clearly as an architecture structured with: Dispatcher, Stores, Actions, Action Creators, and Components (sometimes even with the distinction of View and Controller View). Redux is Flux-inspired, but has significant differences. Maybe we should call it just Flux-inspired architecture. The distinction is important because there are other unidirectional data flow architectures such as in Elm (https://github.com/evancz/...) and Cycle.js (http://cycle.js.org). We might be creating confusion when mentioning "Flux" and meaning different things.

function

function ~~class~~

unnecessary
intermediate-state

```
import {EventEmitter} from 'events';
const TodoStore = Object.assign(
  {}, EventEmitter.propotype
);
TodoStore.dispatchToken =
  MyDispatcher.register(action => {
    switch (action.type) {
    case 'ADD_TODO':
      this._todos.push(action.todo);
      TodoStore.emitChange();
      break;
    case 'DELETE_TODO':
      this._todos.splice(0, action.index);
      TodoStore.emitChange();
      break;
    }
  });
```

ReduceStore

what

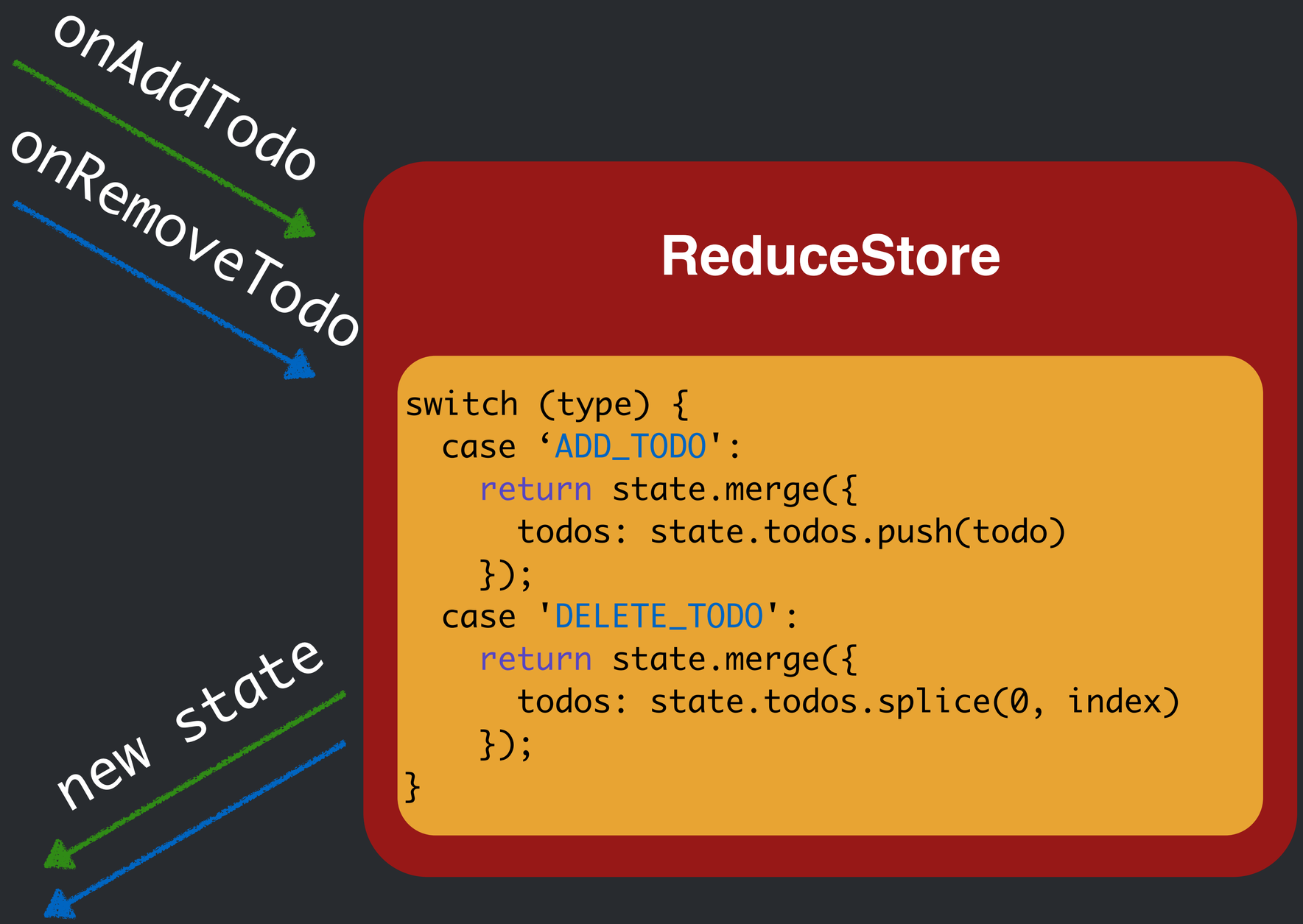
is ReduceStore

ReduceStore

```
switch (type) {
  case ‘ADD_TODO':
    return state.merge({
      todos: state.todos.push(todo)
    });
  case 'DELETE_TODO':
    return state.merge({
      todos: state.todos.splice(0, index)
    });
}
```

ReduceStore

```
switch (type) {
  case ‘ADD_TODO':
    return state.merge({
      todos: state.todos.push(todo)
    });
  case 'DELETE_TODO':
    return state.merge({
      todos: state.todos.splice(0, index)
    });
}
```


onAddTodo
onRemoveTodo
new state
ReduceStore
switch (type) {
case ‘ADD_TODO':
return state.merge({
todos: state.todos.push(todo)
});
case 'DELETE_TODO':
return state.merge({
todos: state.todos.splice(0, index)
});
}

how

to use ReduceStore

```
import {ReduceStore} from 'flux/utils';
class TodoReduceStore extends ReduceStore {
  reduce(state, action) {

  }
}
```

```
import {ReduceStore} from 'flux/utils';
class TodoReduceStore extends ReduceStore {
  reduce(state, action) {
    switch (action.type) {
    case 'ADD_TODO':
      return state.merge({
        isLoading: false,
        todos: state.todos.push(action.todo)
      });









    }
  }
}
```

```
import {ReduceStore} from 'flux/utils';
class TodoReduceStore extends ReduceStore {
  reduce(state, action) {
    switch (action.type) {
    case 'ADD_TODO':
      return state.merge({
        isLoading: false,
        todos: state.todos.push(action.todo)
      });
    case 'DELETE_TODO':
      return state.merge({
        todos: state.todos.delete(action.index)
      });


    }
  }
}
```

```
import {ReduceStore} from 'flux/utils';
class TodoReduceStore extends ReduceStore {
  reduce(state, action) {
    switch (action.type) {
    case 'ADD_TODO':
      return state.merge({
        isLoading: false,
        todos: state.todos.push(action.todo)
      });
    case 'DELETE_TODO':
      return state.merge({
        todos: state.todos.delete(action.index)
      });
    default:
      return state;
    }
  }
}
```

```
import {Container} from 'flux/utils';
class TodoContainer extends Component {
  static getStores() {
    return [TodoReduceStore];
  }
  static calculateState() {
    return TodoReduceStore.getState();
  }
  render() {
    return <Todo {...this.state} />;
 }
}

export default Container.create(TodoContainer);
```

why

ReduceStore

- ease for Flux migration

- ease for Flux migration
- Functional style

Functional *


FOOD GROUPS


FOOD GROUPS

_renderXXX as sub-component

```
class Todo extends Component {
  _renderItem(todo) {
    return <li>{todo.title}</li>
  }
  render() {
    const items = this.props.todos
      .map(this._renderItem);
    return <ul>{items}</ul>;
  }
}
```

Stateless Component

```
function TodoItem(props) {
  return <li>{props.title}</li>
}

class Todo extends Component {
  render() {
    return (
      <ul>
        {this.props.todos.map(TodoItem)}
      </ul>
    );
  }
}
```

why _render is anti-pattern

```
class FooBar extends Component {
  _renderFoo() {
    return <p>foo-{this.props.name}</p>
  }
  _renderBar() {
    return <p>bar-{this.props.name}</p>
  }



}
```

```
class FooBar extends Component {
  _renderFoo() {
    return <p>foo-{this.props.name}</p>
  }
  _renderBar() {
    return <p>bar-{this.props.name}</p>
  }
  render() {
    return this.props.isFoo
      ? this._renderFoo()
      : this._renderBar()
  }
}
```

preserved state

preserved state

difficult to bail out

```
function Foo(props) {
  return <p>foo-{props.name}</p>
}
function Bar(props) {
  return <p>bar-{props.name}</p>
}
```

```
function Foo(props) {
  return <p>foo-{props.name}</p>
}
function Bar(props) {
  return <p>bar-{props.name}</p>
}
class FooBar extends Component {
  render() {
    return this.props.isFoo
      ? <Foo name={this.props.name} />
      : <Bar name={this.props.name} />
  }
}
```


todo
add a todo
hello
world


todo
add a todo
hello
world

```
class TodoItem extends Component {
  render() {
    const { isCompleted, content } = this.props;
    var className = classNames({
      'is-completed': isCompleted
    })
    return (
      <li className={className}>









      </li>
    );
  }
}
```

```
class TodoItem extends Component {
  render() {
    const { isCompleted, content } = this.props;
    var className = classNames({
      'is-completed': isCompleted
    })
    return (
      <li className={className}>
        <input
          className="toggle"
          type="checkbox"
          checked={isCompleted}
          onChange={this._onChange}
        />









      </li>
    );
  }
}
```

```
class TodoItem extends Component {
  render() {
    const { isCompleted, content } = this.props;
    var className = classNames({
      'is-completed': isCompleted
    })
    return (
      <li className={className}>
        <input
          className="toggle"
          type="checkbox"
          checked={isCompleted}
          onChange={this._onChange}
        />
        <label>{content}</label>




      </li>
    );
  }
}
```

```
class TodoItem extends Component {
  render() {
    const { isCompleted, content } = this.props;
    var className = classNames({
      'is-completed': isCompleted
    })
    return (
      <li className={className}>
        <input
          className="toggle"
          type="checkbox"
          checked={isCompleted}
          onChange={this._onChange}
        />
        <label>{content}</label>
        <button
          className="delete-todo"
          onClick={this._onDelete}
        />
      </li>
    );
  }
}
```

```
function TodoInput(props) {
  return (
    <input
      className="toggle"
      type="checkbox"
      checked={props.checked}
      onChange={props.onChange}
    />
  );
}
```

```
function TodoInput(props) {
  return (
    <input
      className="toggle"
      type="checkbox"
      checked={props.checked}
      onChange={props.onChange}
    />
  );
}
function TodoContent(props) {
  return (
    <label>{props.content}</label>
  );
}
```

```
function TodoInput(props) {
  return (
    <input
      className="toggle"
      type="checkbox"
      checked={props.checked}
      onChange={props.onChange}
    />
  );
}
function TodoContent(props) {
  return (
    <label>{props.content}</label>
  );
}
function TodoDeleteButton(props) {
  return (
    <button
      className="delete-todo"
      onClick={props.onDelete}
    />
  );
}
```

```
class TodoItem extends Component {
  render() {
    const { isCompleted, content } = this.props;
    var className = classNames({
      'is-completed': isCompleted
    })
    return (
      <li className={className}>
        <TodoInput
          checked={isCompleted}
          onChange={this._onChange}
        />
        <TodoContent content={content} />
        <DeleteButton onDelete={this._onDelete} />
      </li>
    );
  }
}
```

Decorator/HOC

```
function BaseTodoItem(props) {
  return {
    render() {
      return (
        <label>{props.content}</label>
      );
    }
  };
}
```

```
function MakeCompletable(Child) {
  class Completable extends Component {
    render() {
      const {checked, onChange, ...childProps} = this.props;
      return (
        <div>
          <input
            className="toggle"
            type="checkbox"
            checked={checked}
            onChange={onChange}
          />
          <Child {...childProps} />
        </div>
      );
    }
  }
  return Completable;
}
```

```
function MakeDeletable(Child) {
  class Deletable extends Component {
    render() {
      const {onDelete, ...otherProps} = this.props;
      return (
        <div>
          <Child {...otherProps} />
          <button
            className="delete-todo"
            onClick={onDelete}
          />
        </div>
      );
    }
  }
  return Deletable;
}
```

```
const CompletableItem = MakeCompletable(BaseTodoItem);
const DeletableItem = MakeDeletable(CompletableItem);

class TodoItem extends Component {
  render() {
    const { isCompleted, content } = this.props;
    const className = classNames({
      'is-completed': isCompleted
    });
    return (
      <li className={className}>
        <DeletableItem
          checked={isCompleted}
          content={content}
          onChange={this._onChange}
          onDelete={this._onDelete}
        />
      </li>
    );
  }
}
```

map / filter / reduce

map

```
var todos = [];
this.props.todos.forEach(todo => {
  todos.push({
    title: todo.title,
    isCompleted: todo.isCompleted,
  });
});
return todos;
```

```
var todos = [];
this.props.todos.forEach(todo => {
  todos.push({
    title: todo.title,
    isCompleted: todo.isCompleted,
  });
});
return todos;
```

```
return this.props.todos
  .map(todo => {
    const {title, isCompleted} = todo;
    return {title, isCompleted};
  });
```

filter

```
var todos = [];
this.props.todos.forEach(todo => {
  if (!todo.isCompleted) {
    todos.push(todo);
  }
});
return todos;
```

```
var todos = [];
this.props.todos.forEach(todo => {
  if (!todo.isCompleted) {
    todos.push(todo);
  }
});
return todos;
```

```
return this.props.todos
  .filter(todo => !todo.isCompleted);
```

reduce

```
var todos = {};
this.props.todos.forEach(todo => {
  todos[todo.id] = todo;
});
return todos;
```

```
var todos = {};
this.props.todos.forEach(todo => {
  todos[todo.id] = todo;
});
return todos;
```

```
this.props.todos.reduce((todos, todo) => {
  todos[todo.id] = todo;
  return todos;
}, {});
```

```
// find all incomplete tasks
// where title end with `today`
let todos = [];
```

```
// find all incomplete tasks
// where title end with `today`
let todos = [];
this.props.todos.forEach(todo => {









});
```

```
// find all incomplete tasks
// where title end with `today`
let todos = [];
this.props.todos.forEach(todo => {
  if (
    !todo.isCompleted &&
    todo.title.endsWith('today')
  ) {




  }
});
```

```
// find all incomplete tasks
// where title end with `today`
let todos = [];
this.props.todos.forEach(todo => {
  if (
    !todo.isCompleted &&
    todo.title.endsWith('today')
  ) {
    const {id, title, isCompleted} = todo;
    todos[id] = {
      title,
      isCompleted,
    };
  }
});
```

```
// find all incomplete tasks
// where title end with `today`
let todos = [];
this.props.todos.forEach(todo => {
  if (
    !todo.isCompleted &&
    todo.title.endsWith('today')
  ) {
    const {id, title, isCompleted} = todo;
    todos[id] = {
      title,
      isCompleted,
    };
  }
});
return todos;
```

```
// find all incomplete tasks
// where title end with `today`
return this.props.todos
   .filter(todo => !todo.isCompleted)
```

```
// find all incomplete tasks
// where title end with `today`
return this.props.todos
   .filter(todo => !todo.isCompleted)
   .filter(todo => todo.title.endsWith('today'))
```

```
// find all incomplete tasks
// where title end with `today`
return this.props.todos
    .filter(todo => !todo.isCompleted)
    .filter(todo => todo.title.endsWith('today'))
    .reduce((todos, todo) => {
      const {id, title, isCompleted} = todo;
      todos[id] = {title, isCompleted};
      return todos;
    }, {});
```

why array

```
function process(number) {
  console.log('[number] %s', number);
}

const zero = [];
zero.forEach(process);

const one = [42];
one.forEach(process);

const multiple = [1, 2, 3, 4];
multiple.forEach(process);
```


GitHub, Inc. [US]
https://github.com/ReactiveX/learnrx


A-TYPE
LINES- 000
TOP
010000
SCORE
000000
STATISTICS
NEXT
LEVEL
01
gifbin.com


A-TYPE
LINES- 000
TOP
010000
SCORE
000000
STATISTICS
NEXT
LEVEL
01
gifbin.com

<Questions />