

React 開發實戰



課程大綱

- Redux 簡介
- Redux 核心觀念
 - action / Action Creator
 - reducer
 - store
 - middleware
- React-Redux
 - Provider
 - connect
- Development Tool
- React-Router
- Third party library





Redux 簡介

Redux 簡介

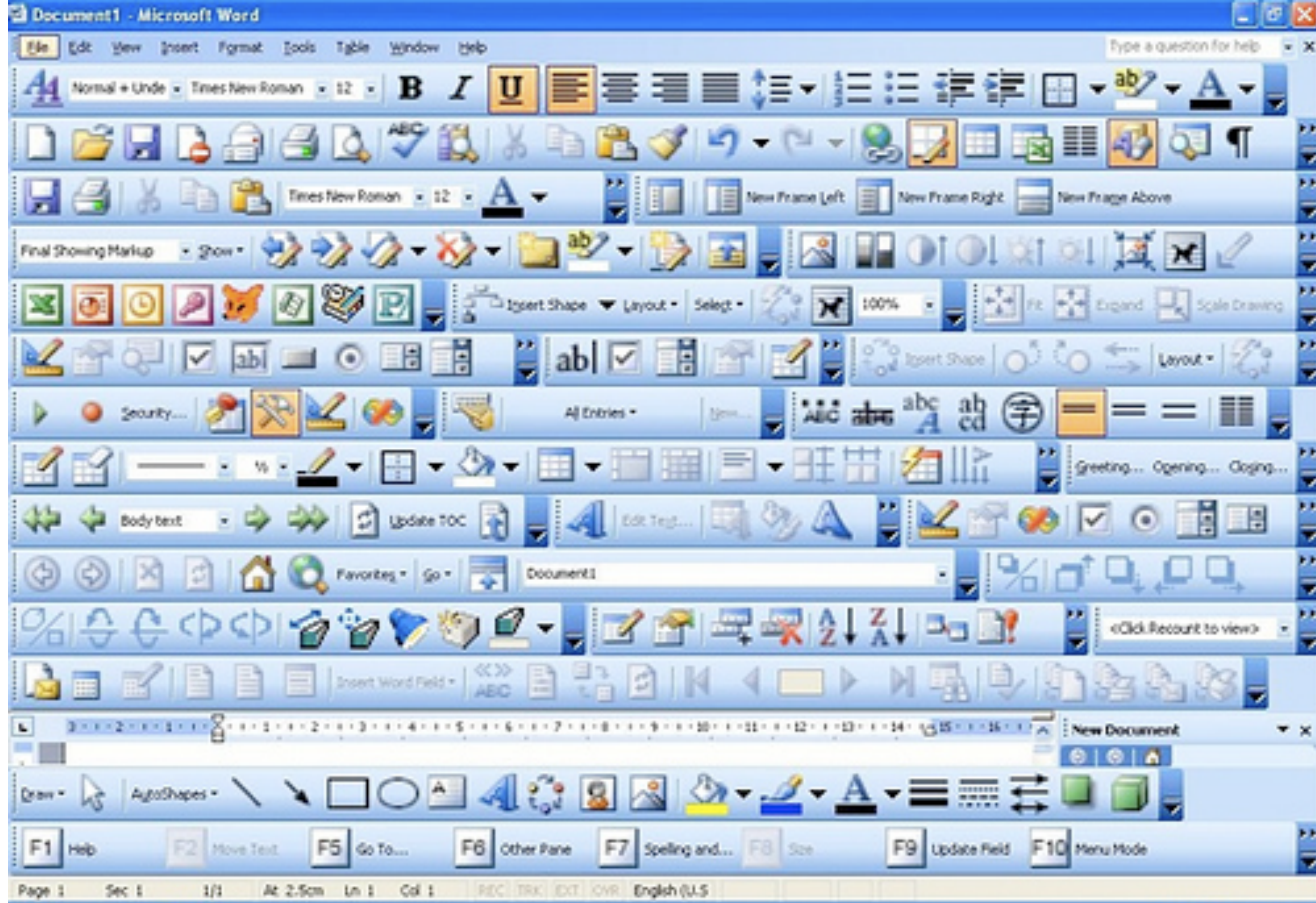
- Redux 是一套用來管理狀態 (state management) 的 Library
- Redux 可以搭配任何前端框架使用，不限於 React
- Redux 是目前 React 生態圈最多人使用的狀態管理工具



為什麼需要 Redux?

- 隨著 Single-page Applications 的功能越多，我們就需要管理更多的狀態，包含來自 Server 的 response 還有快取的資料。
- 要管理每個狀態的變更是非常困難的，當專案變得複雜，很容易喪失對某些資料改變的發生時機、為什麼發生以及改變了哪些狀態的控制





為什麼需要 Redux?

- 隨著 Single-page Applications 的功能越多，我們就需要管理更多的狀態，包含來自 Server 的 response 還有快取的資料。
- 要管理每個狀態的變更是非常困難的，當專案變得複雜，很容易喪失對某些資料改變的發生時機、為什麼發生以及改變了哪些狀態的控制
- 簡單來說，我們需要一個可預測的狀態管理工具。



Redux 的優勢與特色

- 可預測 Predictable
- 集中化 Centralized
- 可除錯 Debuggable
- 彈性 Flexible





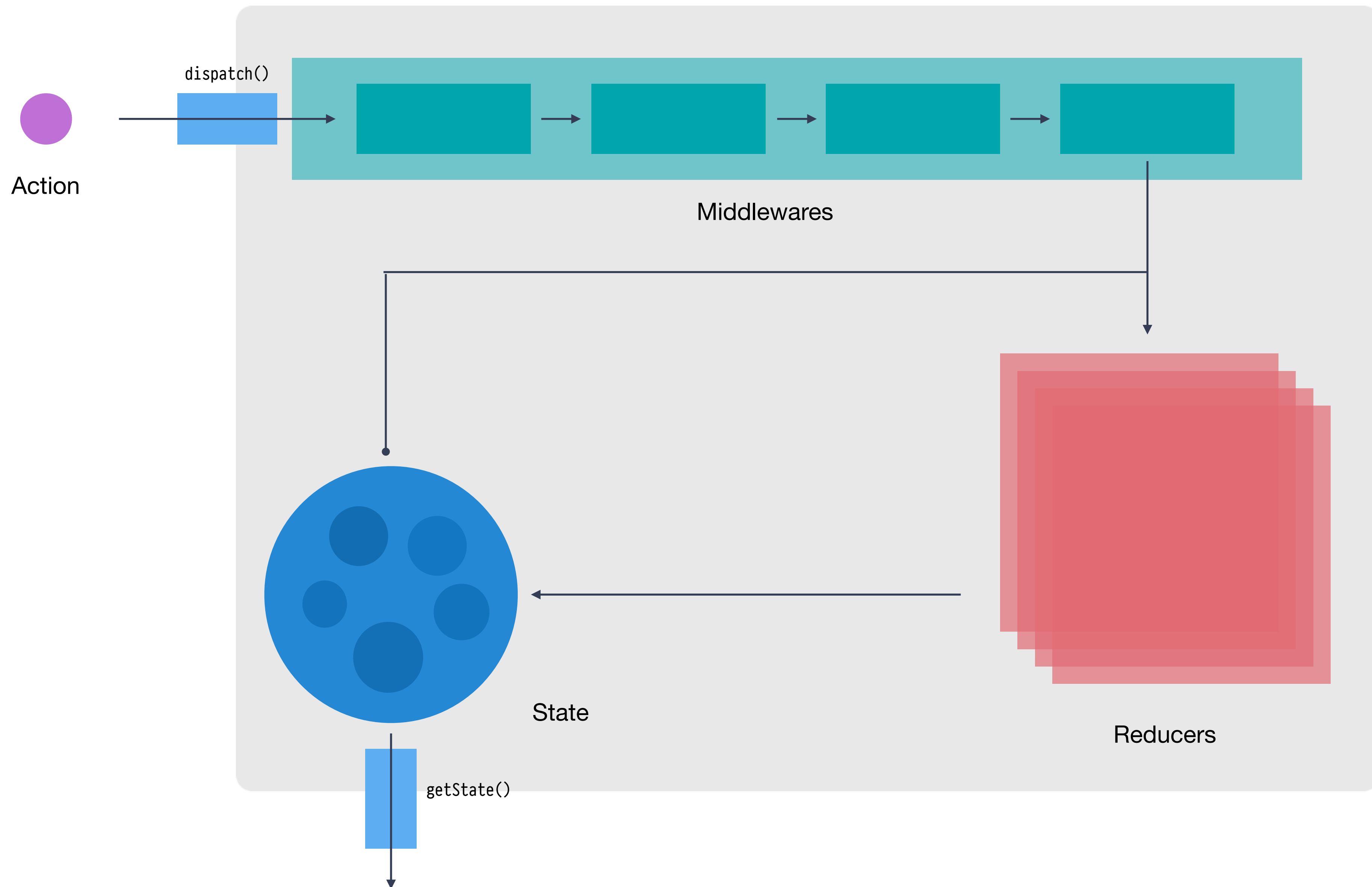
Redux 核心觀念

Redux 三個核心

- Store
 - Reducer
 - Middleware
- Action



Store



reduce

```
[1, 2, 3, 4]
```

```
.reduce(( state, action ) => state + action, 0)
```



reducer

`(state = 0, action) => state + action`



reducer

- reducer 就是一個 pure function
- reducer 吃兩個參數，第一個參數是當前的狀態，第二個參數是 action
- reducer 回傳一個新狀態

`(state = initialState, action) => newState`



reducer

- reducer 就是一個 pure function
- reducer 吃兩個參數，第一個參數是當前的狀態，第二個參數是 action
- reducer 回傳一個新狀態

```
function counter(state = 0, action) {  
  switch(action.type){  
    case 'ADD_COUNT':  
      return state + action.payload  
    default:  
      return state;  
  }  
}
```



action

- action 就是一個 plain object
- 一個具有 type 屬性的 object

```
{  
  type: 'ADD_COUNT'  
}
```



action creator

- 一個 function 用來建立並回傳 action

```
function addCount(payload = 1) {  
  return {  
    type: 'ADD_COUNT',  
    payload  
  };  
}
```



Store

- 由 reducer(與 middleware) 所組成

```
function counter(state = 0, action) {  
  switch(action.type){  
    case 'ADD_COUNT':  
      return state + 1  
    default:  
      return state;  
  }  
}
```

```
const store = createStore(counter)
```



Store

- 由 reducer(與 middleware) 所組成
- 多個 reducer 可以透過 combineReducers 組成一個 root reducer

```
function counter(state = 0, action) {  
  // ...  
}
```

```
function user(state = {}, action) {  
  // ...  
}
```

```
const rootReducer = combineReducers({  
  counter,  
  user  
})  
const store = createStore(rootReducer)
```



Store

- 由 reducer(與 middleware) 所組成
- 多個 reducer 可以透過 combineReducers 組成一個 root reducer
- Store 負責儲存狀態，可以透過 getState() 拿到當前的狀態

```
const state = store.getState()
```



Store

- 由 reducer(與 middleware) 所組成
- 多個 reducer 可以透過 combineReducers 組成一個 root reducer
- Store 負責儲存狀態，可以透過 getState() 拿到當前的狀態
- 可透過 dispatch 傳入 action 來修改當前的狀態

```
var state = store.getState();
```

```
store.dispatch(action);
```

```
var newState = store.getState();
```



Redux - 練習一



Edit on CodeSandbox



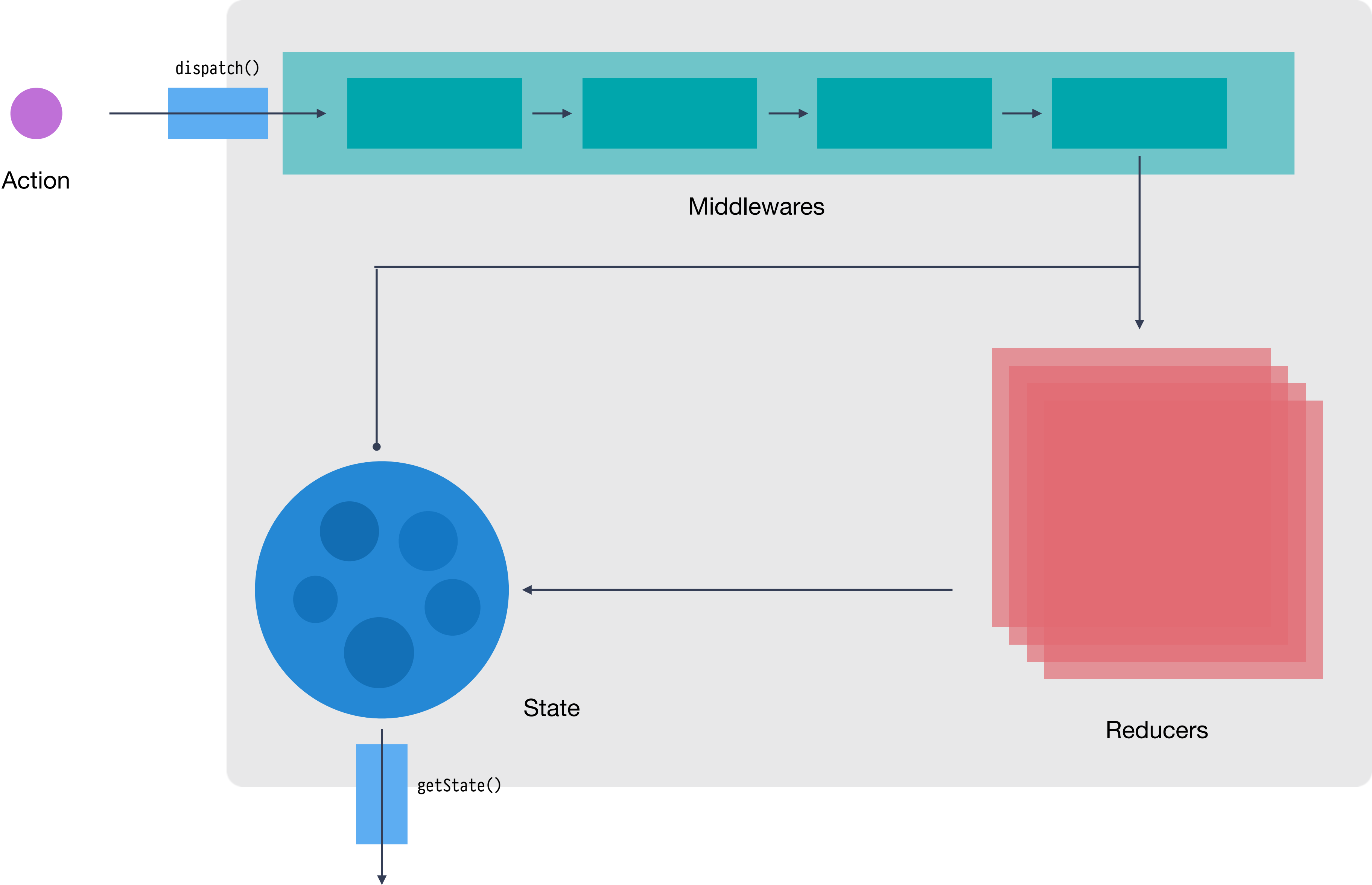
Redux - 練習二



Edit on CodeSandbox



Store



middleware

- 一個 higher order function
- 專門拿來處理 side effect
- 大多數的情境下不用自己寫 middleware

```
const middleware =  
  store => next => action => any
```



middleware

- 一個 higher order function
- 專門拿來處理 side effect
- 大多數的情境下不用自己寫 middleware

```
const logger = store => next => action => {  
  console.log('dispatching', action)  
  let result = next(action)  
  console.log('next state', store.getState())  
  return result  
}
```



Store

加入 middleware

```
const store = createStore(  
  rootReducer,  
  // applyMiddleware() tells createStore() how to  
  applyMiddleware(logger)  
)
```



Redux - 練習三



Edit on CodeSandbox



redux-thunk

- [GitHub](#)
- thunk middleware for redux
- 專門用來處理非同步的 side effect
- 目前 redux 主流的 side effect 處理工具

```
npm install redux-thunk
```

```
import thunk from 'redux-thunk';
```

```
const store = createStore(  
  rootReducer,  
  applyMiddleware(thunk)  
);
```



什麼是 thunk ?

```
let x = 1 + 2;
```

```
let foo = () => 1 + 2;
```



thunk 版本的 action creator

```
function asyncAddCount(payload = 1, ms = 500) {  
  return function(dispatch) {  
    return delay(ms).then(() => {  
      dispatch({  
        type: 'ADD_COUNT',  
        payload,  
      })  
    })  
  };  
};
```



Redux - 練習四



Edit on CodeSandbox



Redux - 練習五



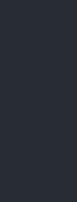
Edit on CodeSandbox



自訂 middleware

Error log collect

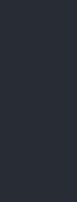
```
const errorLogger = store => next => action => {  
  try {  
    let result = next(action);  
    return result;  
  } catch (err) {  
    console.error(err);  
    logErrorToServer(err)  
  }  
};
```



自訂 middleware

Performance check

```
const benchmark = store => next => action => {  
  var t0 = performance.now();  
  let result = next(action);  
  var t1 = performance.now();  
  console.log(`Call to ${action.type}  
    took ${t1 - t0} milliseconds.`);  
  return result;  
};
```



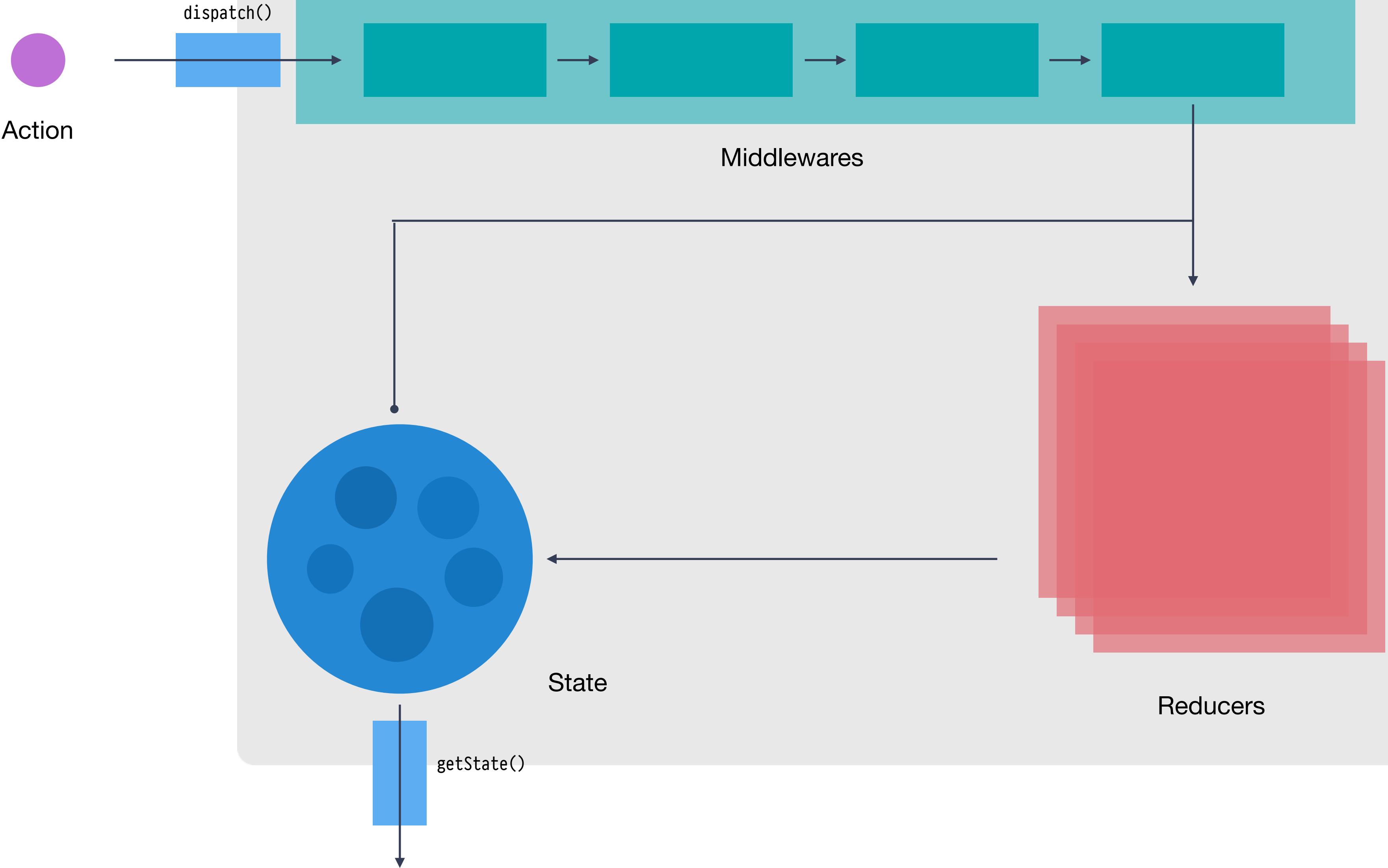
自訂 middleware

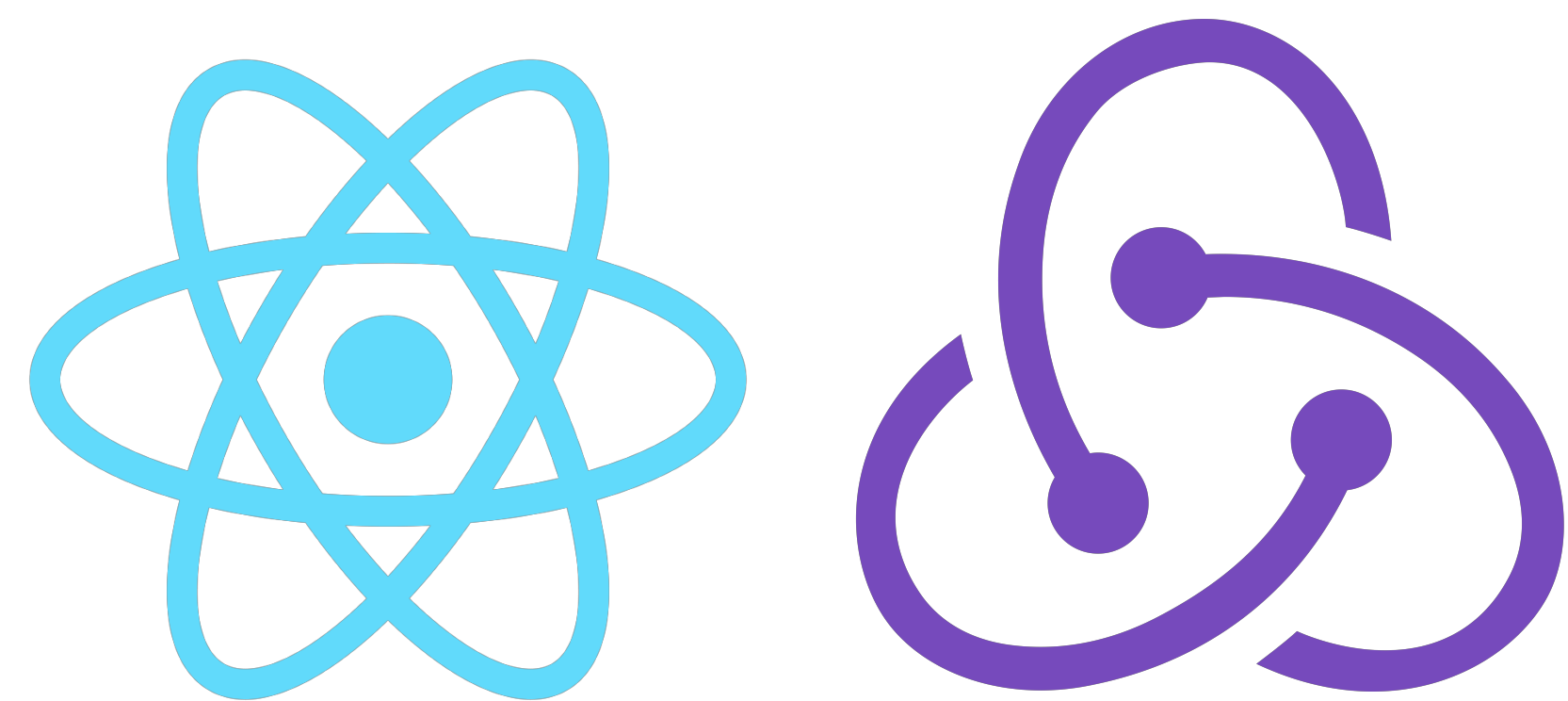
GA tracking

```
const ga = store => next => action => {  
  let result = next(action);  
  ga.send('', action.type, action.payload)  
  return result;  
};
```



Store





React-Redux 快速上手

React-Redux 簡介

- React-Redux 是用來介接 React 跟 Redux 的 Library
- React-Redux 跟 Redux 是完全獨立的 Library
- 其他前端框架也有類似的 Library 跟 Redux 介接
 - 例如：ng-redux, redux-vue



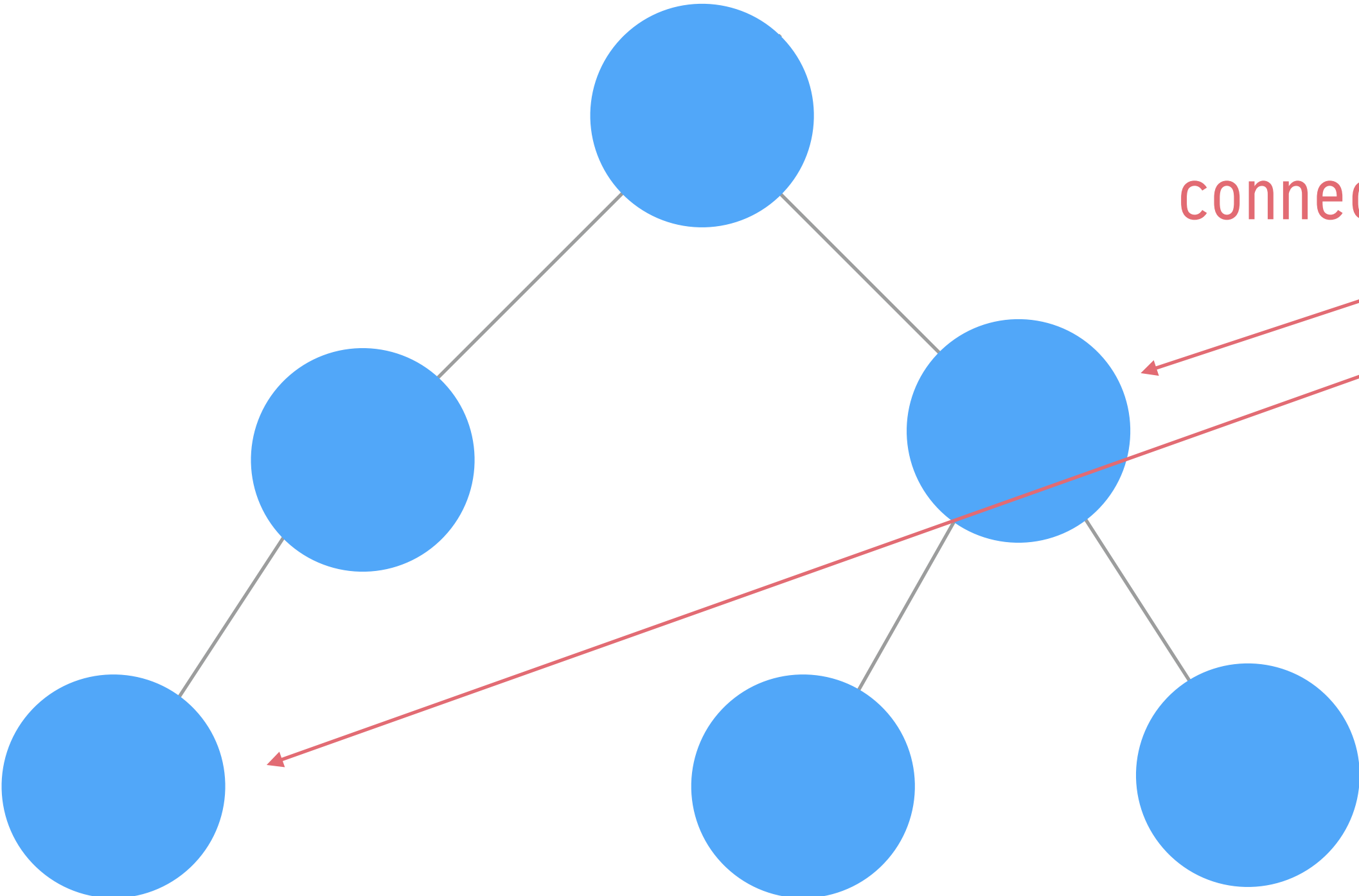
React-Redux 只有一個元件及一個 HOC

- Provider
- connect

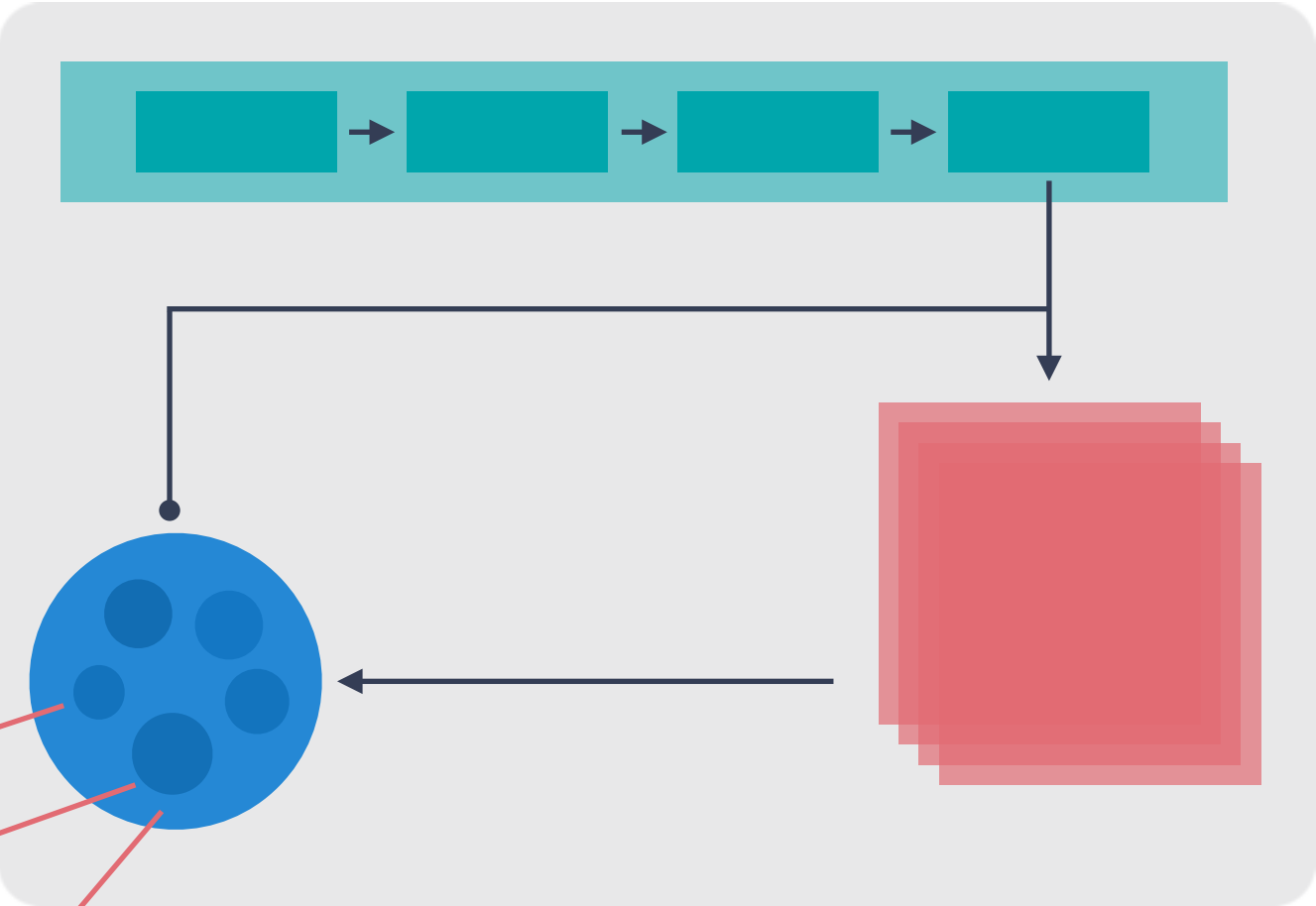


Store

```
<Provider store={store}>
```



connect



安裝 react-redux

```
npm install react-redux
```



Provider

- 一個 React 元件
- 需接收 redux 的 store
- 會在內部透過 react 的 context 傳給子元件

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

```
import { Provider } from 'react-redux'  
import store from './store'
```

```
import App from './App'
```

```
const rootElement = document.getElementById('root')  
ReactDOM.render(  
  <Provider store={store}>  
    <App />  
  </Provider>,  
  rootElement  
)
```



connect

- connect 是一個 Higher order Component
- 第一個參數可用來接收 store 的 state

```
import React from 'react';
import { connect } from 'react-redux'

class App extends React.Component {
  render() {
    const { count } = this.props;

    return <h1>Hello World {count}</h1>;
  }
}

export default (
  connect(state => ({ count: state.count }))(App)
);
```



React-Redux 練習一

安裝 react-redux 並使用 Provider & connect



connect

- connect 是一個 Higher order Component
- 第一個參數用來選擇接收 store 的 state
- 第二個參數用來綁定 store 與 action creator

```
class App extends React.Component {  
  render() {  
    const { count, addCount } = this.props;  
  
    return <h1 onClick={addCount}>  
      Hello World {count}  
    </h1>;  
  }  
}  
  
export default connect(  
  state => ({ count: state.count }),  
  {  
    addCount,  
    asyncAddCount,  
  }  
) (App);
```



React-Redux 練習二

使用 connect 綁定 action creator



React-Redux 練習三

嘗試自己建立 cats reducer 並把 CatPage 的資料改成抓 redux 的資料



Edit on CodeSandbox



React-Redux 練習四

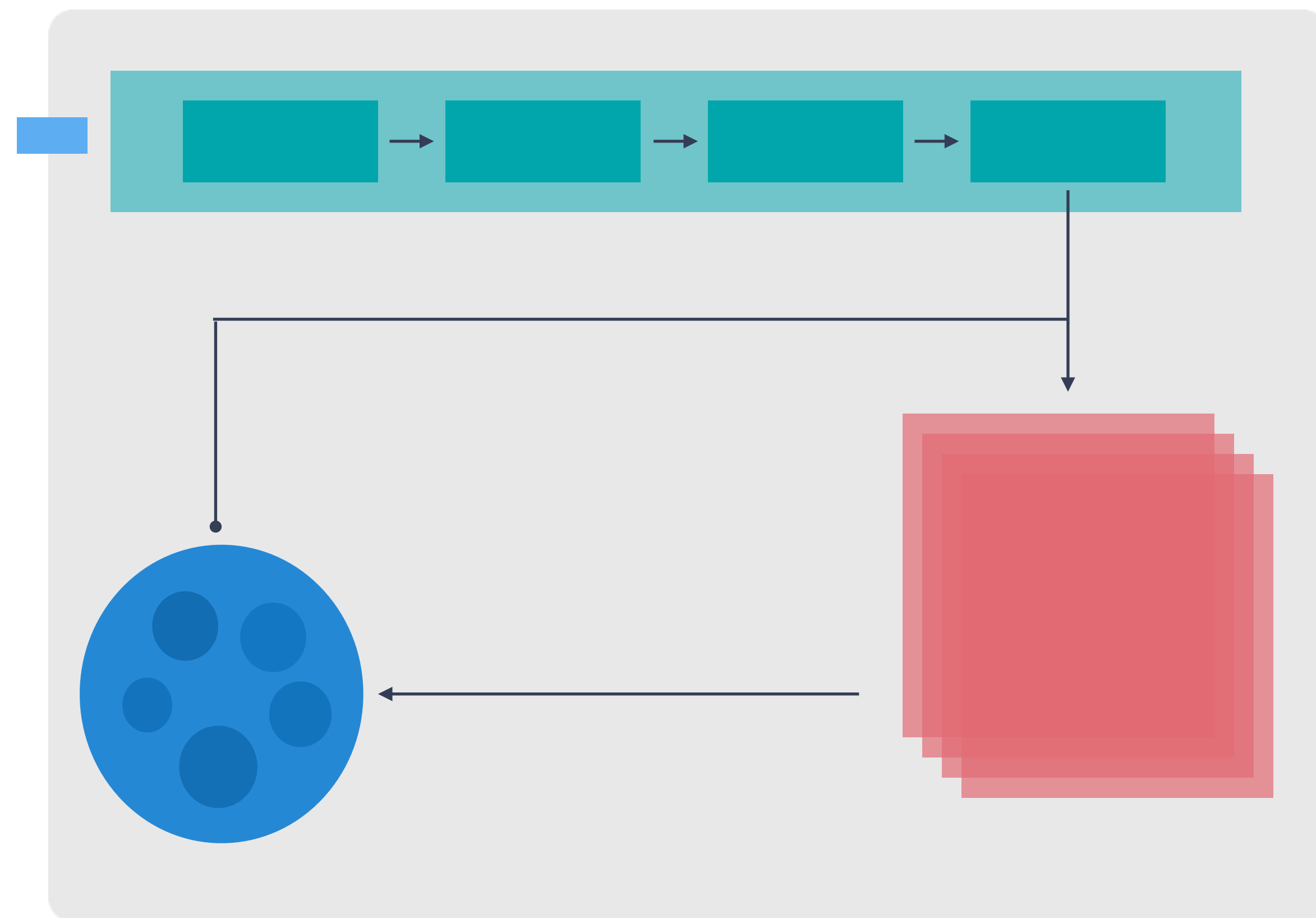
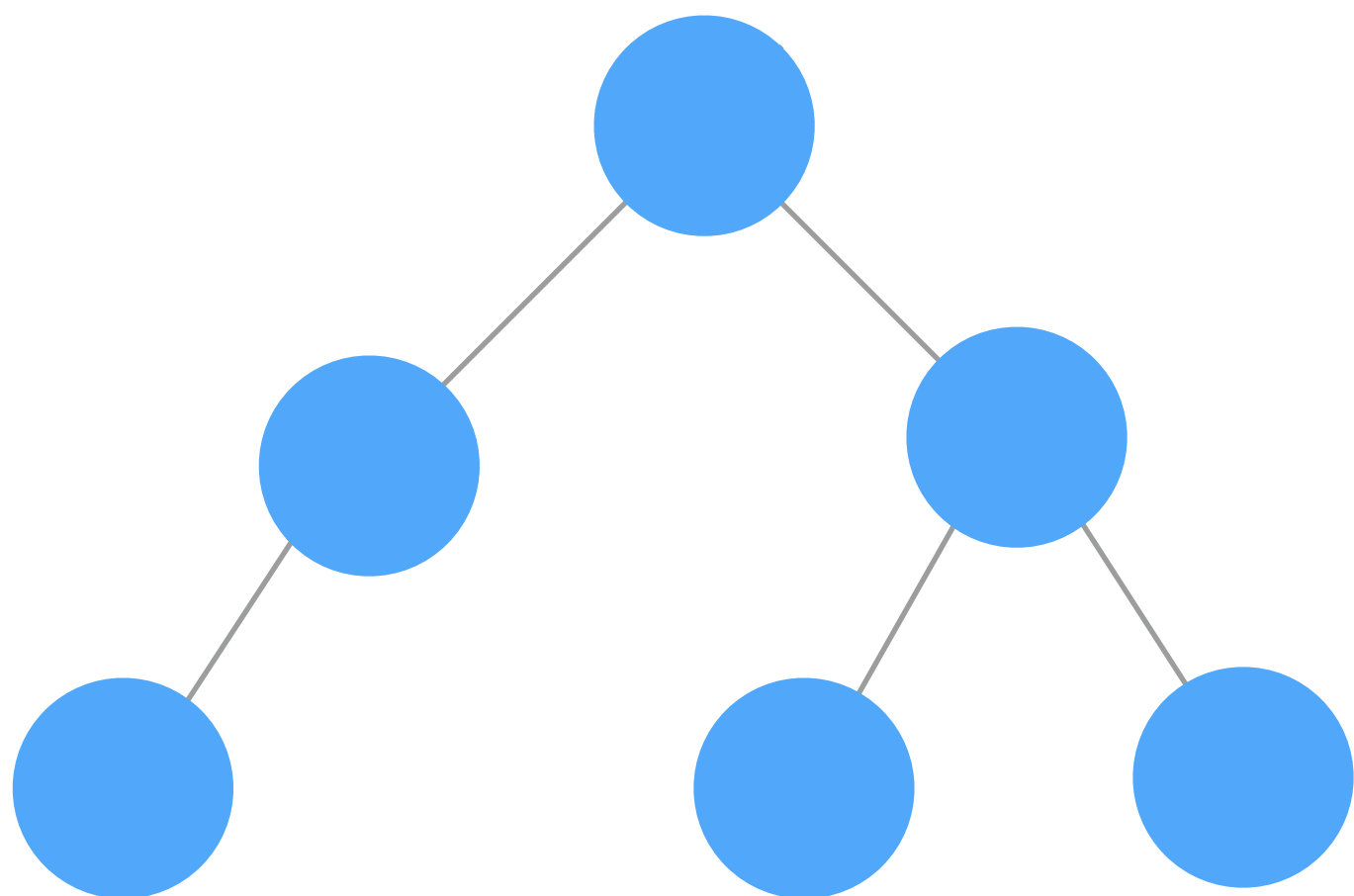
抓 `http://localhost:4000/cats` 的資料



React-Redux 練習五

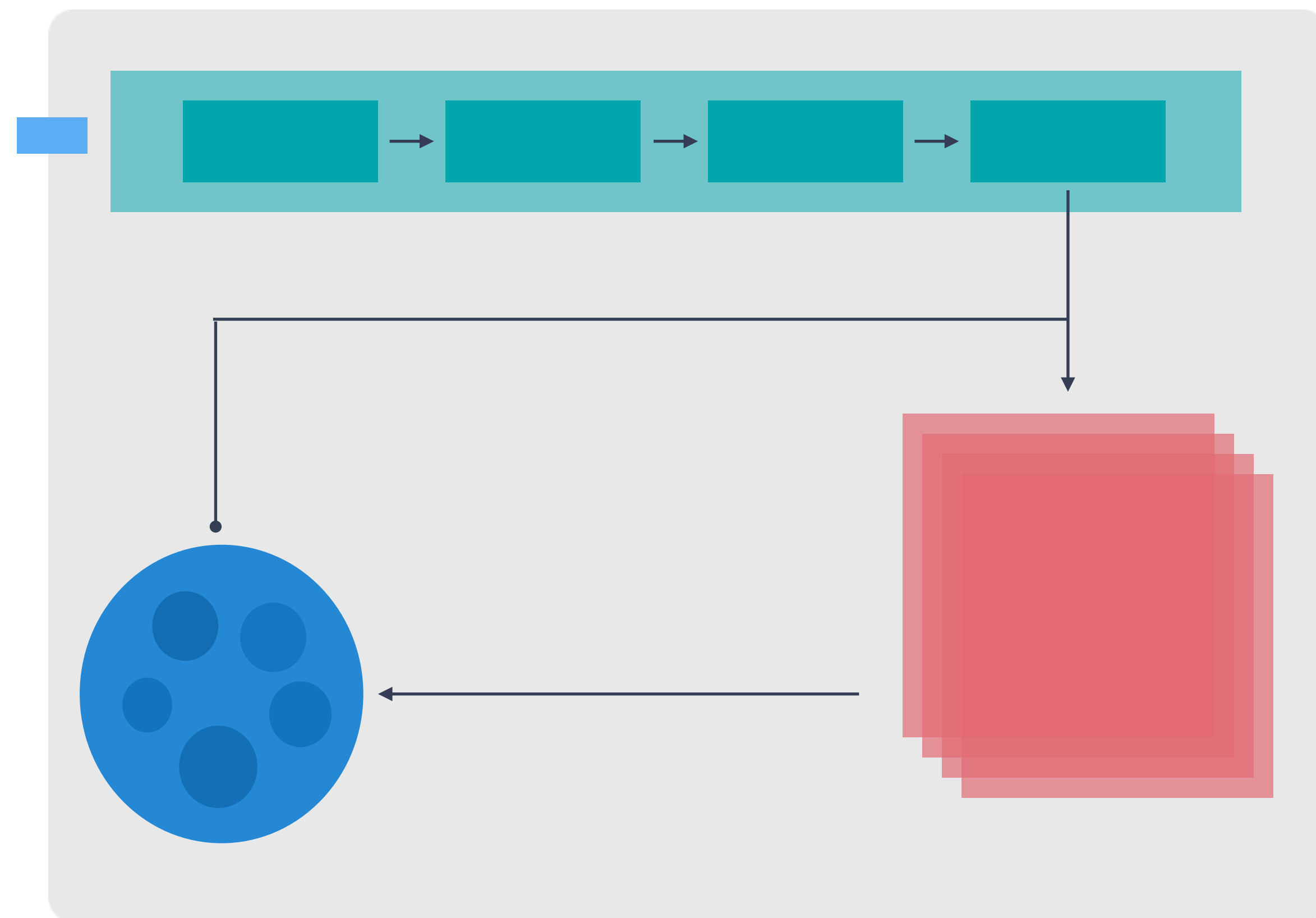
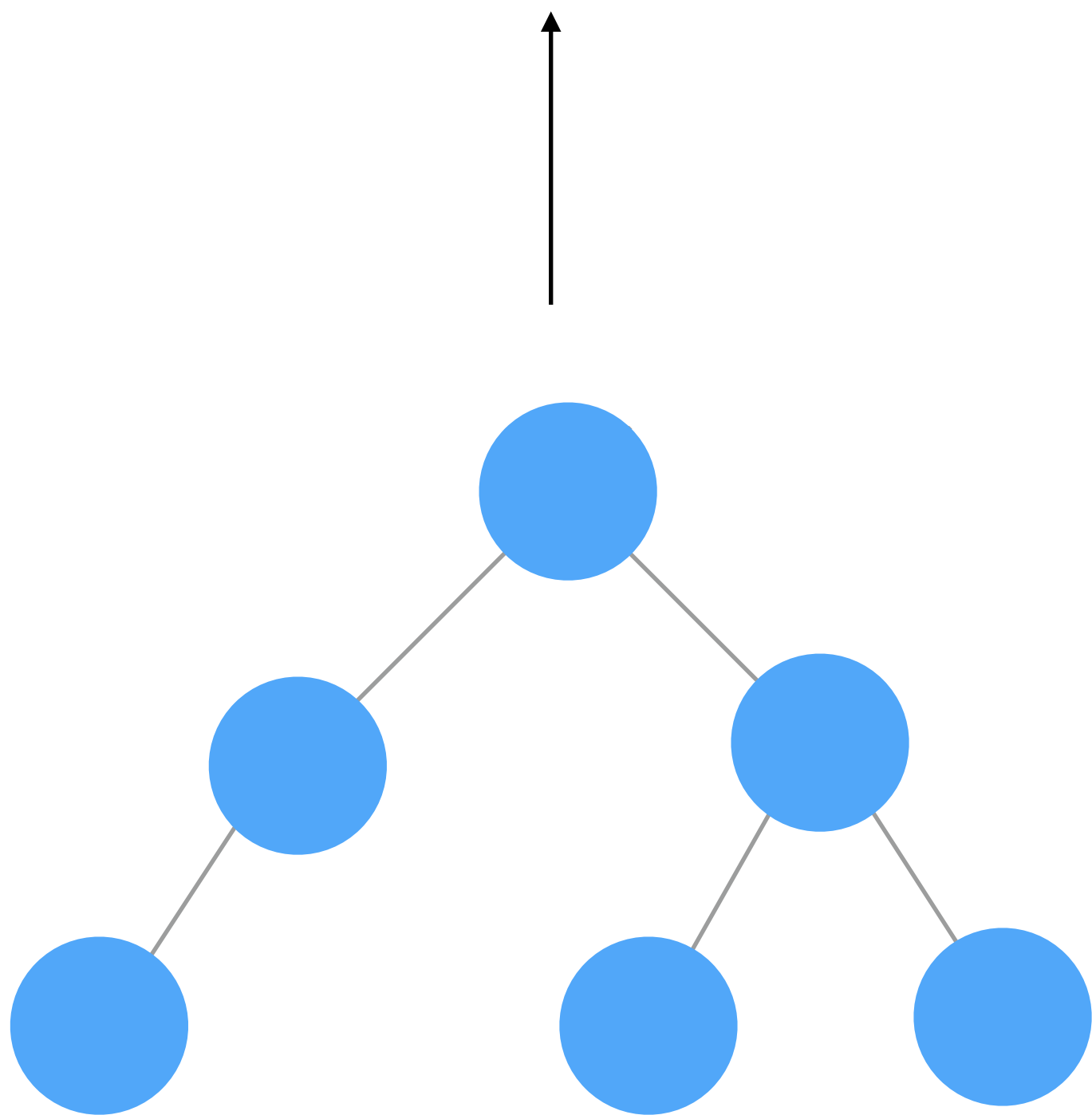
修改 addCat ，打 POST <http://localhost:4000/cats>





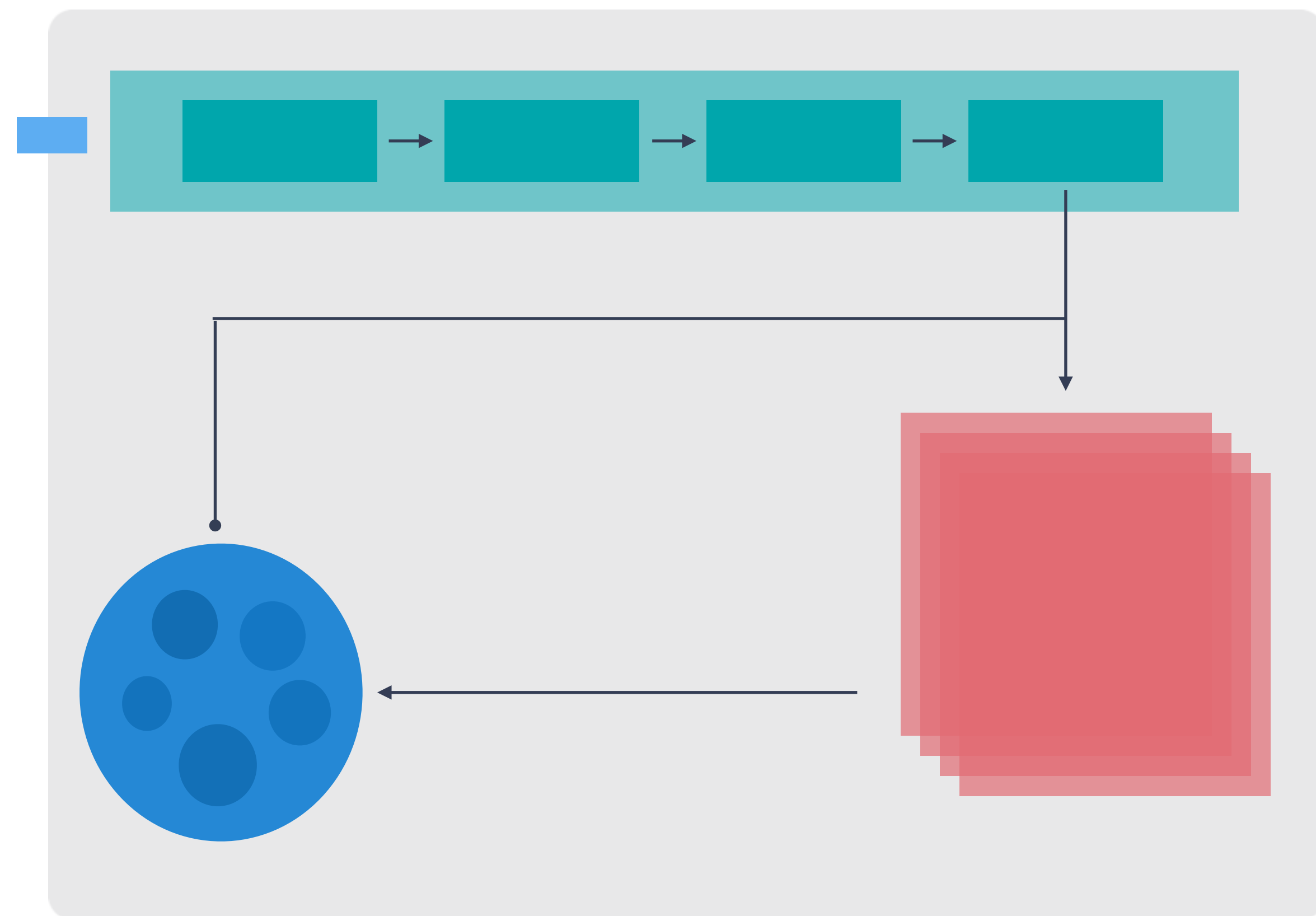
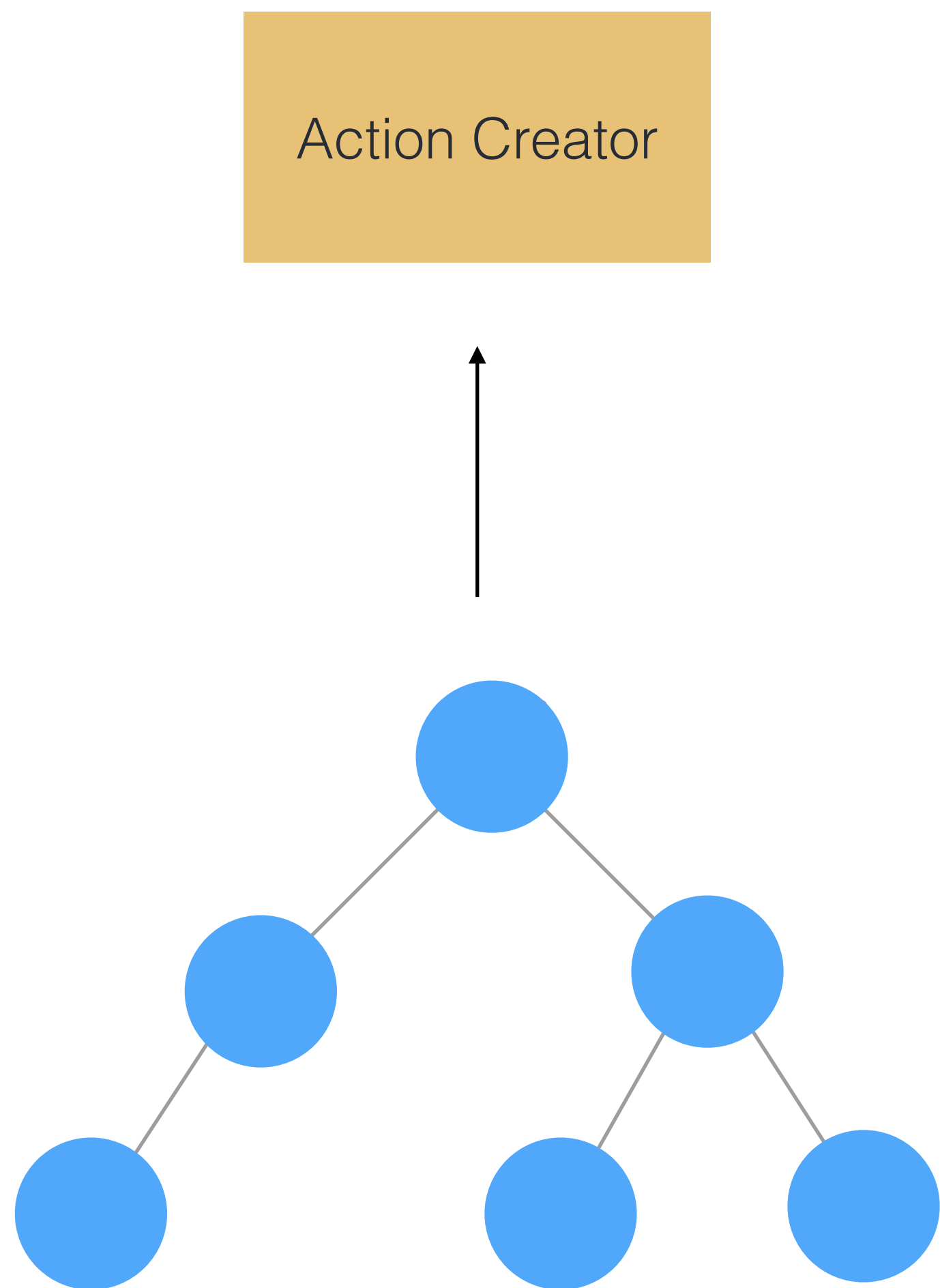
Store





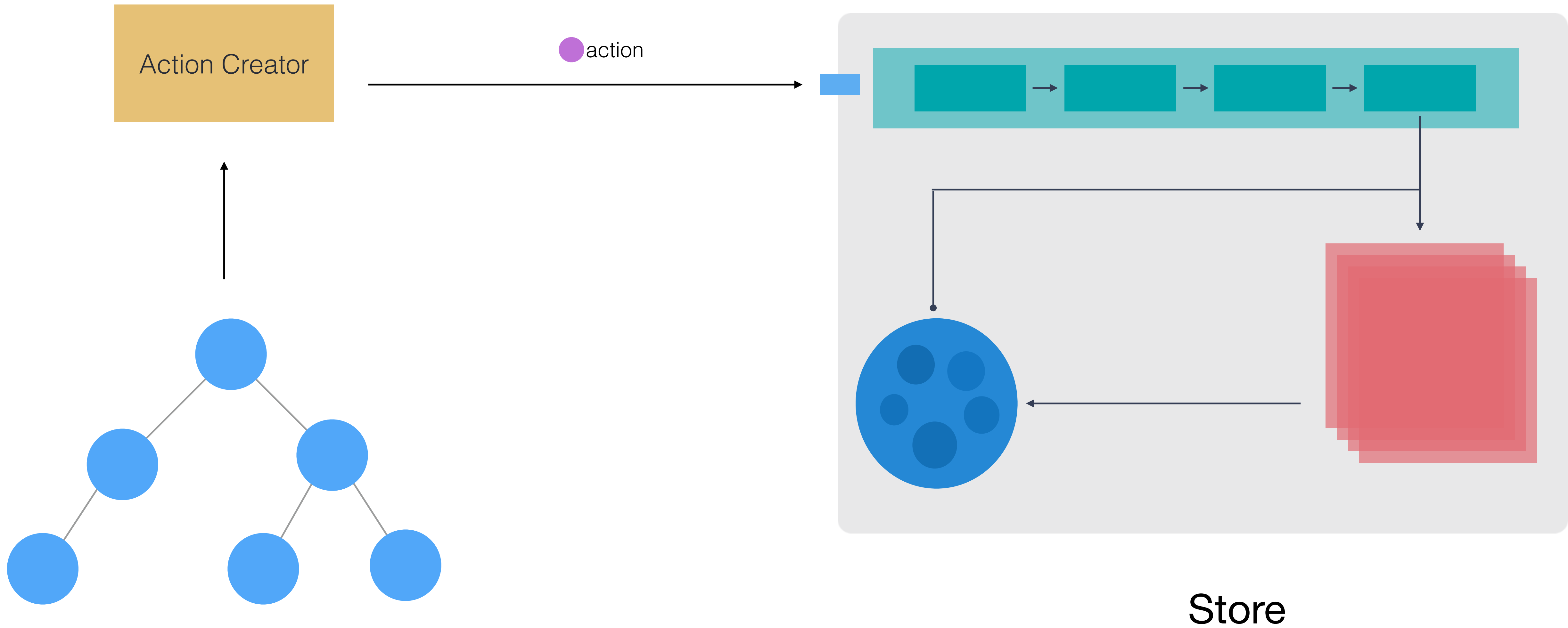
Store

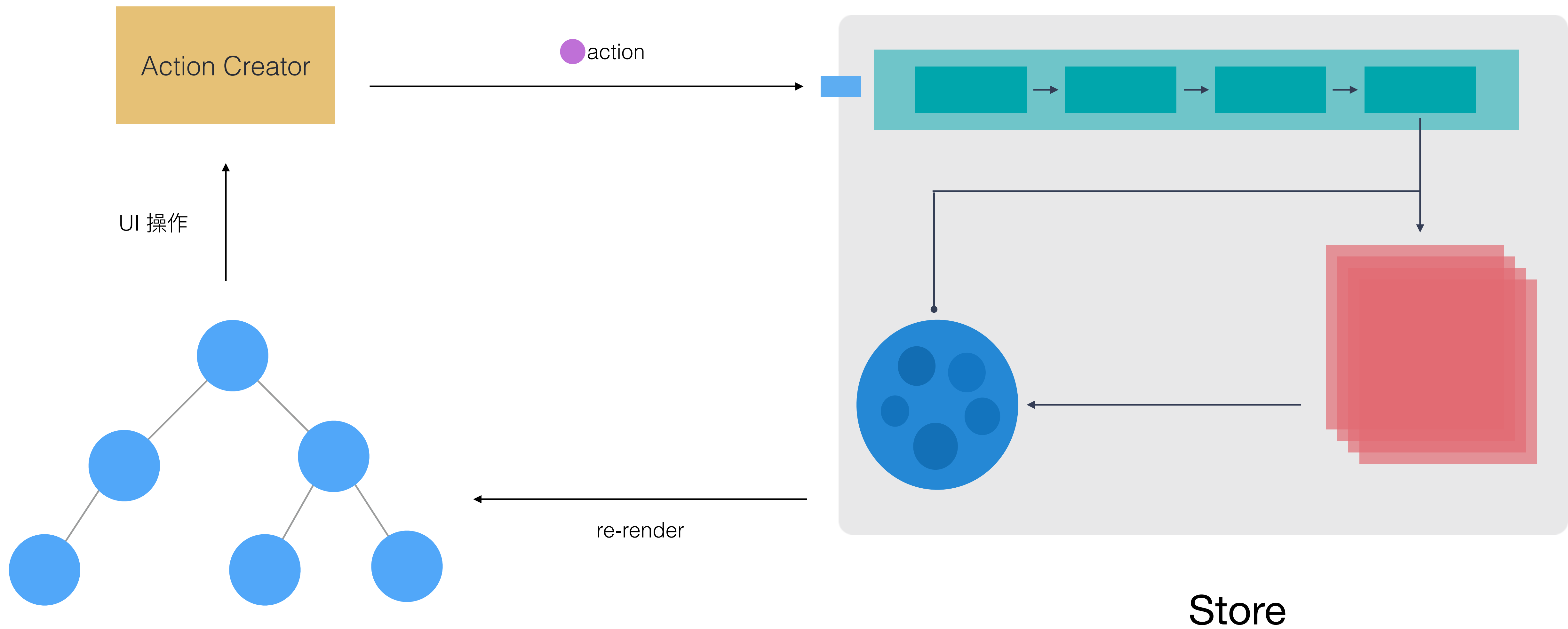




Store

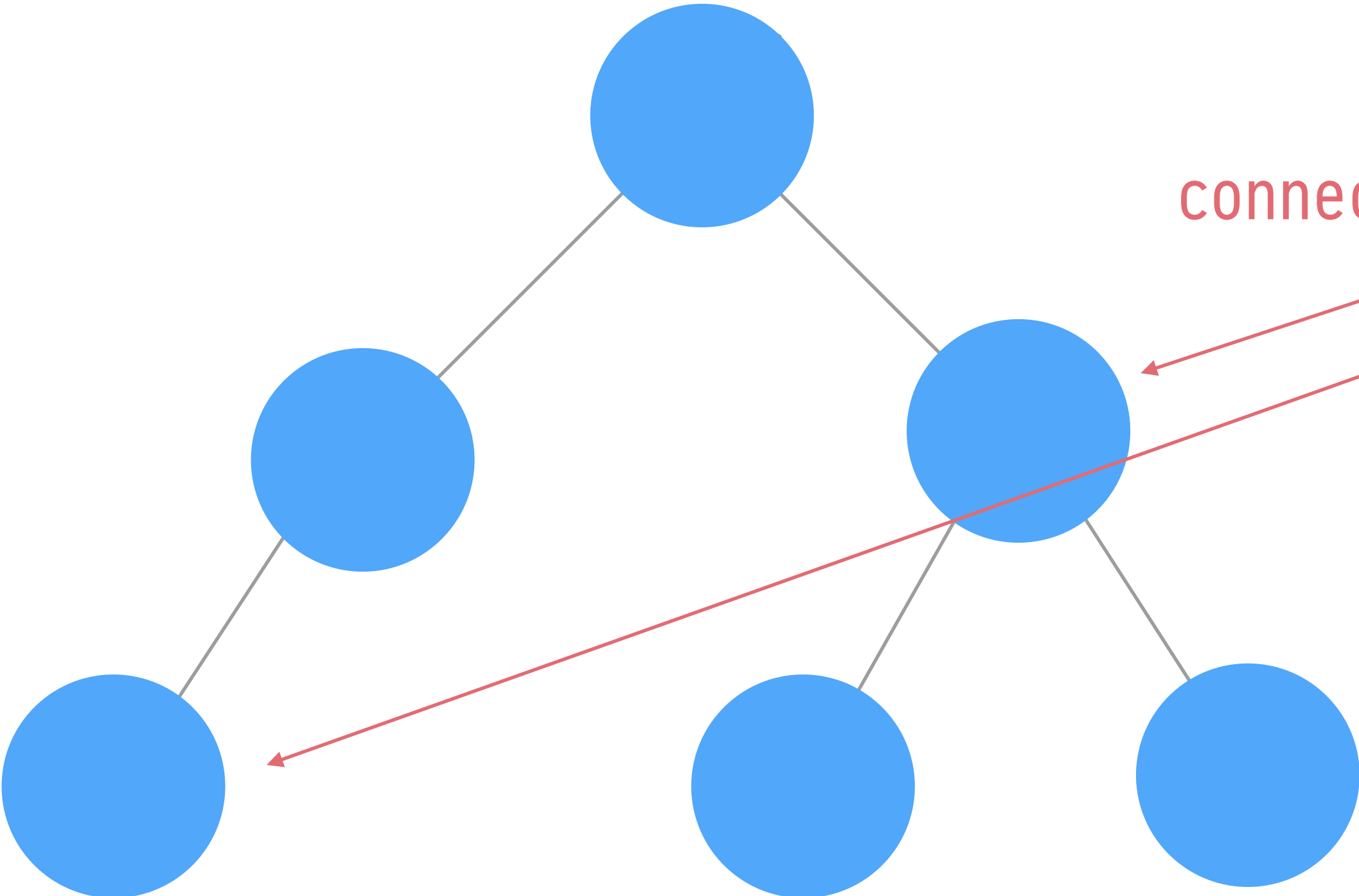




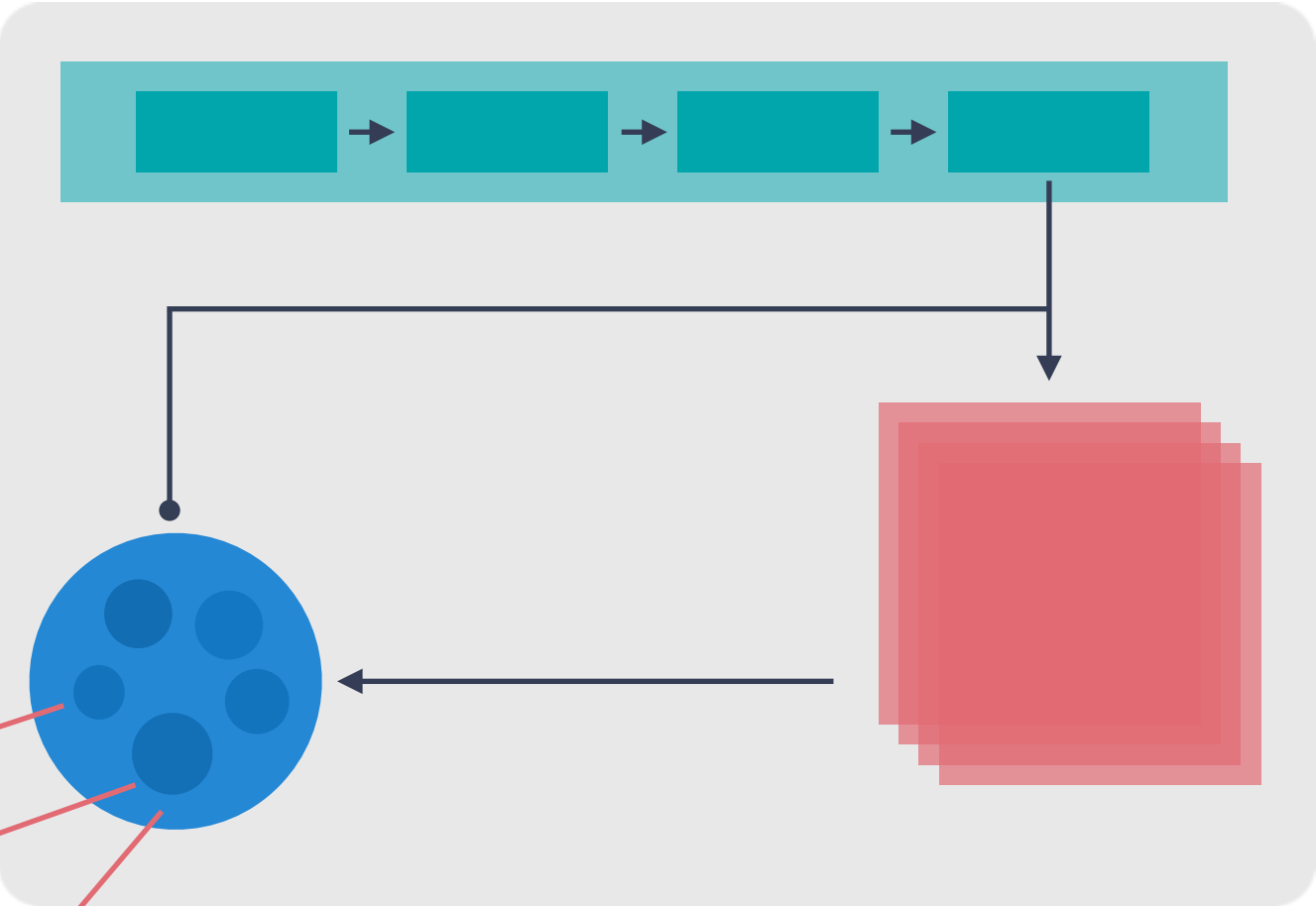


Store

```
<Provider store={store}>
```



connect





DevTools



react-devtools

- 由 FB 官方所推出的開發輔助工具
- 可以查看各個元件拿到的 prop、當前的 state 以及 render 的頻率
- 不需額外撰寫任何程式碼
- 相關連結
 - [Github](#)
 - [Chrome Store](#)





```
▼ <Provider>
  ▼ <Context.Provider>
    ▼ <Connect(App)>
      ▼ <Context.Consumer>
        ▼ <App count={1700}> == $r
          ▼ <div>
            ▼ <h1>
              "Hello World "
              "1700"
            </h1>
            ► <Connect(CatPage)>...</Connect(CatPage)>
          </div>
        </App>
      </Context.Consumer>
    </Connect(App)>
  </Context.Provider>
</Provider>
```

Props

```
► addCount: fn()
  count: 1700
```

redux-devtools-extension

- 最早由 Redux 作者 Dan 實作了 redux-devtools，後來社群把 redux-devtools 做成 chrome extension
- 可以查看所有 action 送出的紀錄、當前的狀態以及時光旅行
- 相關連結
 - [Github](#)
 - [Chrome Store](#)



Inspector

filter...

Commit

@@INIT	6:56:56.53
GET_CAT_SUCCESS	+04:58.87
GET_CAT_REQUEST	+00:03.34
GET_CAT_SUCCESS	+00:00.15
CHANGE_FILTER_STR	+00:07.23
CHANGE_FILTER_STR	+00:00.17
CHANGE_FILTER_STR	+00:00.42
CHANGE_FILTER_STR	+00:00.77
CHANGE_FILTER_STR	+00:02.04
CHANGE_FILTER_STR	+00:00.19
CHANGE_FILTER_STR	+00:04.14
CHANGE_FILTER_STR	+00:00.34
CHANGE_FILTER_STR	+00:00.35
CHANGE_FILTER_STR	+00:00.21
CHANGE_FILTER_STR	+00:00.20
CHANGE_FILTER_STR	+00:11.96
CHANGE_FILTER_STR	+00:00.60
CHANGE_FILTER_STR	+00:00.62
CHANGE_FILTER_STR	+00:00.37
CHANGE_FILTER_STR	+00:00.50
CHANGE_FILTER_STR	+00:00.50

React App

Diff

ActionStateDiffTraceTest

TreeRaw

▼ cats (pin)

filterStr (pin): '111' => '1111'

▶

<>

1x

Pause recording

Lock changes

Persist

Dispatcher

Slider

Import

Export

Remote

Settings

redux-devtools-extension setup

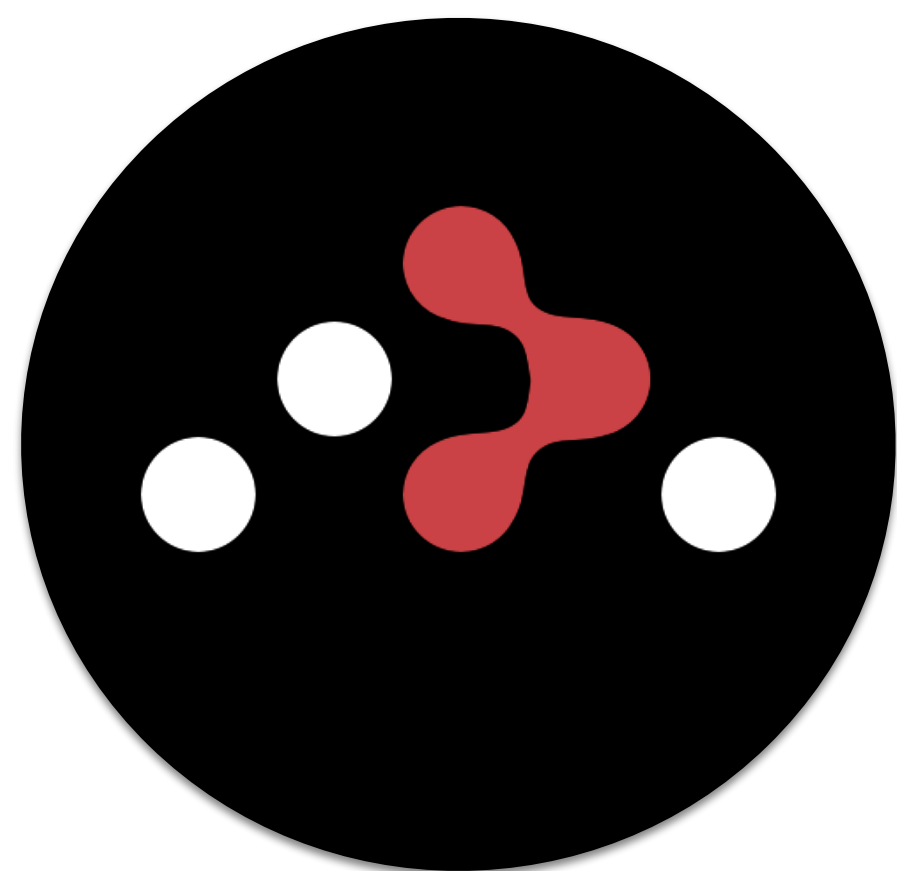


Edit on CodeSandbox

```
npm install redux-devtools-extension
```

```
import {  
  composeWithDevTools  
} from 'redux-devtools-extension';  
  
const store = createStore(  
  rootReducer,  
  composeWithDevTools(applyMiddleware(thunk))  
);
```





React-Router 快速上手

安裝 react-router

```
npm install react-router-dom
```



React-Router 四個基本元件

- Router
- Route
- Switch
- Link



Router

- Router 是一個 React 元件，會負責記錄 History，全站只會使用一個 Router
- 需搭配 Route 使用
- Router 又分為 BrowserRouter、HashRouter 以及 MemoryRouter

```
import {  
  BrowserRouter as Router,  
  Route  
} from "react-router-dom";  
  
<Router>  
  <Route exact path="/" component={Home} />  
  <Route path="/about" component={About} />  
  <Route path="/topics" component={Topics} />  
</Router>
```



Route

- Route 是一個 React 元件，由外部傳入的 path 決定是否要 render 元件
- exact 為 true 時，path 做完全比對。
- 若沒有搭配 Switch 使用會比對全部的 Route，只要比對中的都會顯示
- 需搭配 Router 使用

```
import {  
  BrowserRouter as Router,  
  Route  
} from "react-router-dom";  
  
<Router>  
  <Route exact path="/" component={Home} />  
  <Route path="/about" component={About} />  
  <Route path="/topics" component={Topics} />  
</Router>
```



Route Path

- url 參數
- `/:variableName`

```
<Route path="/:id" component={Child} />

function Child(props) {
  return (
    <div>
      <h3>ID: {props.match.params.id}</h3>
    </div>
  );
}
```



Route Path

- url 參數規則
 - /users/:id
 - /users/1 match
 - /users not match
 - /users/:id?
 - /users/1 match
 - /users match

```
{/*
```

It's possible to use regular expressions to control what param values should be matched.

```
* "/order/asc" - matched
```

```
* "/order/desc" - matched
```

```
* "/order/foo" - not matched
```

```
*/}
```

```
<Route
```

```
  path="/order/:direction(asc|desc)"
```

```
  component={ComponentWithRegex}
```

```
/>
```



Switch

- Switch 是一個 React 元件
- Switch 內的 Route 只比對到第一個，後面的就不顯示

```
import {
  BrowserRouter as Router,
  Route,
  Switch
} from 'react-router-dom';

<Router>
  <Switch>
    <Route exact path="/" component={Home} />
    <Route path="/about" component={About} />
    <Route path="/:user" component={User} />
    <Route component={NoMatch} />
  </Switch>
</Router>;
```



Link

- Link 是一個 React 元件
- 負責做導覽，一定要傳入 to 屬性

```
import { Link } from 'react-router-dom'
```

```
<Link to="/about">About</Link>
```



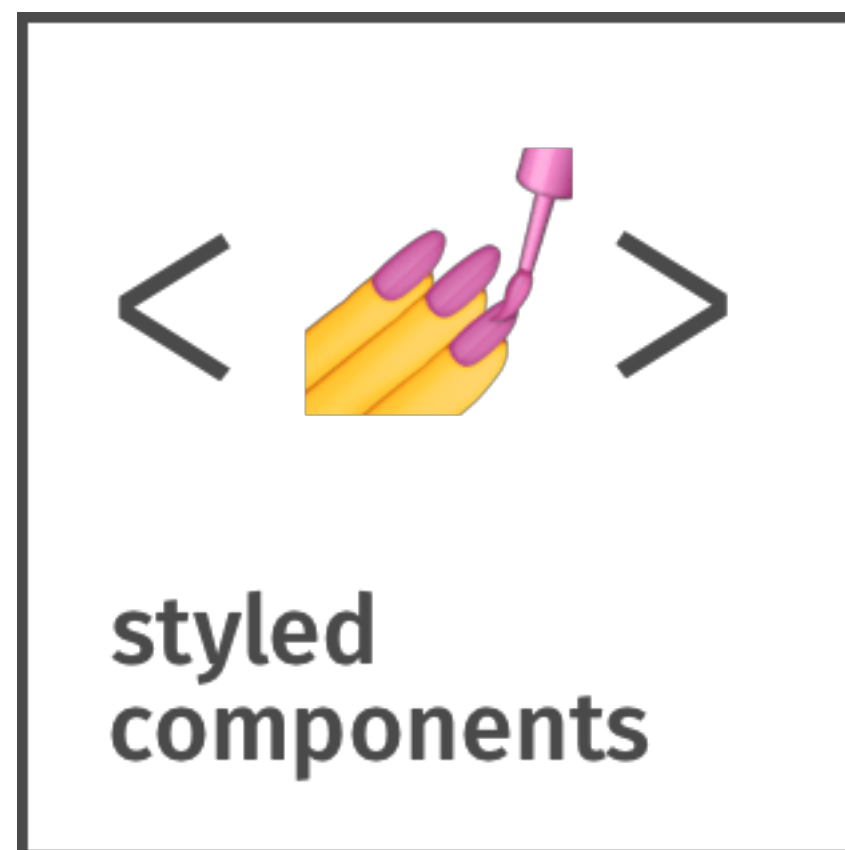
React-Router 練習



Edit on CodeSandbox



Third Party Library

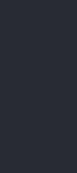


Kendo UI[®]
by  Telerik

安裝

styled-components

```
npm install styled-components
```



styled-components

- 一套 CSS-in-JS 的 Library
- 用來解決 CSS Naming Scope 的問題
- 使用 ES6 的 Tagged templates



Edit on CodeSandbox

```
const Button = styled.a`
  /* This renders the buttons above... Edit me! */
  display: inline-block;
  border-radius: 3px;
  padding: 0.5rem 0;
  margin: 0.5rem 1rem;
  width: 11rem;
  background: transparent;
  color: white;
  border: 2px solid white;

  /* The GitHub button is a primary button
   * edit this to target it specifically! */
  ${props => props.primary && css`
    background: white;
    color: palevioletred;
  `}
`
```

,



安裝 polish

```
npm install polish
```



polish

- 一套輕量的 toolset for styling
- 可搭配 styled-components 使用

```
import { lighten, modularScale } from 'polished'

const div = styled.div`
  background: ${lighten(0.2, '#FFCD64')};
  background: ${lighten('0.2', 'rgba(204,205,100,0.7)')};
`
```



安裝

@progress/kendo-scheduler-react-wrapper

```
npm install @progress/kendo-scheduler-react-wrapper
```



kendo-scheduler-react-wrapper

- 由 jQuery 包裝成 React 的套件
- 目前已被棄用，不在 kendo-react 中
- 文件網址：<https://www.telerik.com/kendo-react-ui/wrappers/scheduler/>

```
<Scheduler
  height={600}
  views={this.views}
  dataSource={this.dataSource}
  date={new Date("2013/6/13")}
  startTime={this.startTime}
  resources={this.resources}
/>
```



安裝

@progress/kendo-treeview-react-wrapper

```
npm install @progress/kendo-treeview-react-wrapper
```



@progress/kendo-treeview-react-wrapper

- 由 jQuery 包裝成 React 的套件
- 目前已被棄用，不在 kendo-react 中
- 文件網址：<https://www.telerik.com/kendo-react-ui/wrappers/treeview/>

```
<TreeView  
  checkboxes={this.checkboxes}  
  check={this.onCheck}  
  dataSource={this.dataSource}  
>;
```

