L'ENVIRONNEMENT RRRR

REACT + RAMDA + REDUX + RESELECT

confoo_mtl_2017

Benjamin Dreux @ code3.ca

REACT

- librairie de vue
- Incite au fonctionnel
- JSX
- Pas de gestion de l'état
- Pas de communication serveur

```
const redButton = ({label, onClick}) => {
  return <button style="backgroundColor: red" onClick={onClick}>
    {label}
  </button>
}
```

REDUX

- Conçu pour React
- Gestion centralisé l'état
- Update(state) => rendu de l'app au complet
- Possibilité de couper des branches de rendu
- État ~ une grosse map

```
const reducer = (state, action) => {
  switch(action.type) {
    case "ADD_CONTACT":
      return addContact(state, action)
  }
}
```

```
const component = ({name, age}) => {...}

const stateToProps = (state) => {
  return {
    name: state.name,
    age: nbYearSince(state.birthdate, new Date())
  }
}
const dispatchToProps = (dispatch) => {
  return {
    onClick: _ => dispatch({type:"gotoPage", page:"profile" })
  }
}
export default connect(stateToProps, dispatchToProps)(component)
```

RAMDA

- Manipulation de donnée
- Type de donnée natif (Map, Array, String)
- Non-destructive
- API très fonctionnel (map, flatten, reduce)
- Structure toujours dernier paramètre (!= lodash)
- Currifié

Manipulation de base

```
const m = {k: "v"}
const n = R.assoc("j", "w", m)
//=> n === {k: "v", j: "w"}

const a = [1, 2, 3]
const a2 = R.append(4, a)
//=> a2 === [1, 2, 3, 4]

const str = R.replace("a", "A",  "aBC")
// => str === "ABC"
```

Manipulation avancée

```
const m = {cmpt: 10, l: [1, 2], str: "abc"}

const update = {cmpt: R.inc, l: R.append(3)}

R.evolve(update, m) // => {cmpt: 11, l: [1, 2, 3], str: "abc"}
```

RESELECT

- Calcul de valeur dérivé
- Memoization
- Composable

MEMOIZATION

```
let count = 0
const factoriel = (n) => {...}
const fact = n => { count++; return factoriel(n) }
const fact_mem = R.memoize(fact)
fact_mem(3) //=> 6
fact_mem(3) //=> 6
count //=> 1
```

```
const state = {
  lines: [
    {desc: "Milk", price: 3.50},
    {desc: "Bread", price: 11.33}
  ],
  taxe: 0.12
};

const priceWT = createSelector((state) => state.lines.price))
const taxe = createSelector((state) => state.taxe)
const taxeAmount = createSelector(priceWT, taxe,
  (priceWT, taxe) => priceWT * taxe
)
const total = createSelector(priceWT, taxAmount,
  (priceWT, taxAmount) => priceWT + taxeAmount
)
```

PROBLÈMES

- Problématiques courantes ou nouvelles

ROUTAGE

```
render((
  <Router history={browserHistory}>
    <Route path="/" component={App}>
      <Route path="about" component={About}/>
      <Route path="users" component={Users}>
        <Route path="/user/:userId" component={User}/>
      </Route>
      <Route path="*" component={NoMatch}/>
    </Route>
  </Router>
), document.getElementById('root'))
```

@from react-router's readme

QUEL COMPOSANT UPDATER ?

AUCUN: *manuellement*

TOUS: *(mentalement)*

QUAND UPDATER UN COMPOSANT ?

Via dépendance:

- Redux.connect
- Reselect.createSelector

QUI DOIT AVOIR ACCÈS AU STORE ?

(presque) TOUS

- Composition de composant plus simple
- Branche d'update coupé rapidement

COMMENT GÉRER LE STORE ?

- Redux fournit un cadre
- Pas de mutation => Ramda
- 1 reducer

Exemple de reaction

```
const reducer = (state, action) => {
  const fns = {
    addItemToCart: addItemToCart
  }
  const fn = fns[action.type] || R.always
  return fn(state, action)
}
```

```
const addItemToCart = (state, action) => {
  const newSubtotal = R.add(action.newItem.price, state.subtotal)
  const newTaxes = newSubtotal * state.taxRate
  const newTotal = newSubtotal + newTaxes
  const tr = {
    items: R.push(action.newItem),
    subtotal: R.always(newSubtotal),
    taxes: R.always(newTaxes),
    total: R.always(newTotal)
  }
  return R.evolve(tr, state)
}
```

AFFICHER UNE IMAGE AVEC UN DÉFAUT EN CAS DE PROBLÈMES

```
currentUrl(){
  if(this.state.imgStatus === "ERROR"|| !this.props.imgSrc) {
    return this.props.defaultSrc
  }
  return this.props.imgSrc
}

render(){
  const currentUrl = this.currentUrl()
  if(["ERROR", "LOADED"].includes(this.state.imgStatus)){
    return <img src={currentUrl}/>
  }
  return <img src={currentUrl}
              onLoad={this.onLoad} onError={this.onError}/>
}
```

AFFICHAGE DE GRANDE LISTE

recherce => [ids]

[ids] => sommaire([ids])

id => détails(id)

Stratégie

```
state = {
  itemsById: {
    42: {
      détails: {...},
      sommaire: {...}
    }
  },
  itemsToRender: [42, 10,...],

}
```

```
//Items sans sommaire:

R.find(
  (id) => R.isNil(state.itemsById[id].sommaire),
  R.keys(state.itemsById)
)

// Items sans détails:
R.find(
  (id) => R.isNil(state.itemsById[id].details),
  R.keys(state.itemsById)
)

// Nouvel item:
(id) => R.not(R.contains(R.keys(state.itemsById), id))
```

AFFICHAGE DE LISTE À RETARDEMENT

```
render(){
  const itemsToRender = R.slice(0,
                                this.props.lastVisibleIndex,
                                this.props.list)
  const completed = R.equals(this.props.lastVisibleIndex,
                             R.length(this.props.list))
  const visibilityChange = shown => shown ? this.loadMore() : null
  return <div>
      {R.map(this.props.renderItem, itemsToRender)}
      {!completed ?
        <span>
          <VisibilitySensor onChange={visibilityChange}/> }
          <LoadMore/>
        </span>: null }
    </div>
}
```

MODIFICATION DE FORMULAIRE

```
const form = ({firstname, lastname, invalidFields, onChange})=> {
  return <form>
      <input type="text" value={firstname} placeholder="firstname"
             onChange={onChange("firstname")}
             className={{invalid: invalidFields["firstname"]}}>

      <input type="text" value={lastname} placeholder="lastname"
             onChange={onChange("lastname")}
             className={{invalid: invalidFields["lastname"]}}>
  </from>
}
```

```
const stateToProps = (state) => {
  return {firstname, lastname, invalidFields} = state
}
const dispatchToProps = (dispatch) => ({
  onChange: (fieldName) => (value) => {
    dispatch({type: "change_field", fieldName, value})
}})
```

```
const onChangeField = (state, action) => {
  const invalidFields = !validate(action.value) ?
    R.push(action.fieldName) :
    R.remove(action.fieldName)

  const tr = {
    invalidFields: invalidFields,
    [action.firstname]: R.always(action.value)
  }
  return R.evolve(tr, state)

}
```

COMPOSITION D'ACTION

```
  export default apiQuery => (dispatch, getState) => {
  const queryMatchingIds = ({apiQuery}, cb) => api.getList(apiQuery, cb)
  const matchs = ({searchResults}, cb) => cb(null, searchResults)
  const getIdsToFetch = ({matchs}, cb) => {
    const isInCache = id => R.path(['items', id], getState())
    const detailsToFetch = R.reject(isInCache, matchs)
    cb(null, detailsToFetch)
  }

  const fetchItems = ({idsToFetch}, cb) => api.getSummaries(idsToFetch,
  const sink = (err, {searchResults}) => {
    const action = err ? {type: "error", err} :
      {type: 'show_items', ids: searchResults}
    dispatch(action)
  }
}
```

```
  auto({
    apiQuery:         (cb) => cb(null, apiQuery)),
    searchResults:   ['apiQuery', queryMatchingIds],
    matchs:          ['searchResults', matchs],
    idsToFetch:      ['matchs', getIdsToFetch],
    estateSummaries: ['idsToFetch', fetchItems],
  }, sink)
```

WHY DID YOU UPDATE ?

```
getDisplayName = o => o.displayName ||
                      o.constructor.displayName ||
                      o.constructor.name
```

```
componentWillUpdate(nextProps, nextState){
  const uselessRender =
    R.equals(nextProps, this.props) &&
    R.equals(nextState, this.state)
  if(uselessRender){
    console.info(`Useless render for ${getDisplayName(this)}`)
    console.info("props", this.props, "nextProps", nextProps)
    console.info("state", this.state, "nextState", nextState)
  }
}
```

@voir github.com/garbles/why-did-you-update

TRADUCTION

```
const TrMessage = () => {

  return  <FormattedMessage
      id="welcome"
      defaultMessage={`Hello {name}, you have {unreadMsg, number} {
        unreadMsg,
        plural,
        one {message}
        other {messages}
      }`}
      values={user}
  />

}
```

@voir React-intl

Extraction automatique via Babel-react-intl

Format JSON

```
[
  {
    "id": "welcome",
    "defaultMessage": "Hello {name}, you have {unreadMsg, number} {unrea
  }
]
```

1 json / composant

quelques coups de (∩ •́ ヮ •̀)⊃━☆ﾟ.*･｡ﾟ

```
{
  "welcome": "Bonjour {name}, vous avez {unreadMsg, number} {unreadMsg,
}
```

1 json / locale

ANIMATION INTERRUPTIBLE

```
class Demo extends Component {
  handleMouseDown() {
    this.setState({open: !this.state.open})
  }
  render() {
    return <div>
      <button onMouseDown={this.handleMouseDown}>Toggle</button>
      <Motion style={{x: spring(this.state.open ? 400 : 0)}}>
        {({x}) =>
          <div className="demo">
            <div className="demo-block" style={{
              transform: `translate3d(${x}px, 0, 0)`
            }} />
          </div>}
      </Motion>
    </div>
  }
}
```

@voir React-Motion