Why React is just better in Reason

Hi!

Ricky Vetter

Front End Engineer
Messenger Web

@rickyvetter everywhere

Today

- React and Reason philosophies
- React better with Reason today
- React better with Reason in the future

Disclaimer

- Only talking about the React function API (as opposed to the class API)

Philosophy

- https://reactjs.org/docs/design-principles.html
- https://ocaml.org/learn/description.html
- https://reasonml.github.io/docs/en/what-and-why
- https://bucklescript.github.io/docs/en/what-why

Preferring Immutable Values

- React encourages immutability of values and referential equality checks to ensure that you get the performance and behavior you want:
  - `useEffect(myFn, [dep1, dep2])`
  - `React.memo`
  - `React.useCallback, React.useMemo`

Preferring Immutable Values

- Reason is immutable by default
  - Common data structures (Records, Variants, Lists)
  - Special data types to model mutability (refs, mutable fields)

Escape Hatches

- refs allow you to model mutation across different renders and event handlers in your component
- useImperativeHandle allows you do model imperative/ object oriented code
- dangerouslySetInnerHTML - the ultimate escape hatch to render exactly what you wants at any point

Escape Hatches

```
let one = ref(1);

one := 2;
```

```
type mutablePerson = {
  mutable firstName: string,
  mutable lastName: string,
};
```

Escape Hatches

```
external iSolemnlySwearThisStringIsAnInt: string => int = "%identity";

let myInt = iSolemnlySwearThisStringIsAnInt("hey");

let myString: string = Obj.magic(myInt);
```

Escape Hatches

```
let add: (int, int) => int = [%raw
  {|
  function(a, b) {
    console.log("hello from raw JavaScript!");
    return a + b
  }
|}
];

Js.log(add(1, 2));
```

Interoperability and gradual adoption

- React was originally designed for iterative and experimental adoption
- `React.render` targets specific nodes to be "owned" by React
- Explicitly designed to not completely control nodes so you can "own" children with IDs, data attributes, `dangerouslySetInnerHTML`, etc

Interoperability and gradual adoption

- BuckleScript, and GenType
- per-file compilation, es modules and require
- bs.* annotations to get idiomatic JS output
- genType annotations to get full type coverage with TypeScript or Flow

Reason is better
today

Quick Context

```
const MyComponent = ({name}) => {
  const [count, setCount] = React.useState(() => 0);

  return (<div>
    <p> {name + " clicked " + count + " times"} </p>
    <button onClick={_ => setCount(_ => count + 1)}>
      Click me
    </button>
  </div>);
};
```

Quick Context

```
[@react.component]
let make = (~name) => {
  let (count, setCount) = React.useState(() => 0);

  <div>
    <p> {React.string(
      name ++ " clicked " ++ string_of_int(count) ++ " times"
    )} </p>
    <button onClick={_ => setCount(_ => count + 1)}>
      {React.string("Click me")}
    </button>
  </div>
};
```

Quick Context

```
[@bs.obj]
external makeProps:
  (~name: 'name, ~key: string=?, unit) => {. "name": 'name} = "";

let make = (Props) => {
  let name = Props##name;
  let (count, setCount) = React.useState(() => 0);

  <div>
    <p> {React.string(
      name ++ " clicked " ++ string_of_int(count) ++ " times"
    )} </p>
    <button onClick={_evt => setCount(count => count + 1)}>
      {React.string("Click me")}
    </button>
  </div>
};
```

Types

- Component boundaries
- Hooks, other top level apis
- TS, Flow, inference

First Class Language Tools

```
let (state, dispatch) =
  React.useReducer(
    (state, action) =>
      switch (action) {
      | Click => {...state, count: state.count + 1}
      | Toggle => {...state, show: !state.show}
      },
    {count: 0, show: true},
  );
```

First Class Language Tools

```
let url = ReasonReactRouter.useUrl();
switch (url.path) {
| ["book", id, "edit"] => handleBookEdit(id)
| ["book", id] => getBook(id)
| ["book", id, _] => noSuchBookOperation()
| [] => showMainPage()
| ["shop"]
| ["shop", "index"] => showShoppingPage()
| ["shop", ...rest] =>
  /* e.g. "shop/cart/10", but let "cart/10" be handled by another function */
  nestedMatch(rest)
| _ => showNotFoundPage()
};
```

Explicit handling of "elements"

```
<div> {React.string("hello")} </div>
<div> "hello" </div>
<div> hello </div>
```

- React.string/array
- Enforces consistent rendering and explicit points of wackiness - "gatekeeper"
- `Objects are not valid as a React child (found: object with keys {foo}).`

Props are separate from the implementation

```
type size =
  | Default
  | Large
  | Small;
[@react.component]
let make =
```

- make vs makeProps
- Allows you to define default props without re-implementation

```
module Large = {
  let make = make;
  let makeProps = makeProps(~size=Large);
};
```

Control over the type of props

```
type person = {
  .
  "firstName": string,
  "lastName": string,
};

let makeProps = (~firstName, ~lastName, ()): person => {
  "firstName": firstName,
  "lastName": lastName,
};
```

```
type person;

let makeProps: (~firstName: string, ~lastName: string, unit) => person;
```

The ReasonReact experience can be even better

createElement -> jsx

- https://github.com/reactjs/rfcs/blob/createelement-rfc/text/0000-create-element-changes.md

- The goal is to bring element creation down to this logic:

```
function jsx(type, props, key) {
  return {
    $$typeof: ReactElementSymbol,
    type,
    key,
    props,
  };
}
```

createElement -> jsx

- ```
  function createElement(
    type,
    props,
    ...children
  ) {
  ```

- ```
  function jsx(
    type,
    props,
    key,
  ) {
  ```

createElement -> jsx

- Keys
- Children
- Refs

createElement -> jsx

- Performance benefits (fewer runtime checks for keys, refs, children)
- Simpler code (no special casing for refs and children, keys become explicitly special cased)
- The types make sense - no access to key inside your component, children are always of type `React.element`

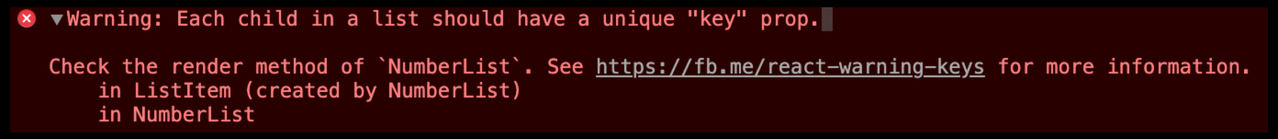
createElement -> jsx

- In Reason this means simplifying how we handle keys, refs and children
- No more magically adding keys and refs to prop types for you
- No reliance on special edge case React behavior to get correct children semantics
- and ...

Key Warnings

- Dynamic list updates
- Helps keep React's work to a minimum

Key Warnings



Key Warnings

```
[@bs.module "react"]
external jsx:
  (component('props), 'props, option(string)) => element
  = "jsx";
```

Key Warnings

```
[@bs.module "react"]
external jsxKeyed:
  (component('props), 'props, string) => element(keyed)
  = "jsx";

[@bs.module "react"]
external jsx:
  (component('props), 'props) => element(unkeyed)
  = "jsx";
```

Key Warnings - Gatekeeper

```
external array:
  array(element(keyed)) => element(unkeyed)
  = "%identity";
```

Key Warnings - Result

```
<div>
  {
    React.array(
      Belt.Array.map(friends, friend =>
        <div> {React.string(friend.firstName)} </div>
      ),
    )
  }
</div>
```

```
38 | {
39 |   React.array(
40 |     Belt.Array.map(friends, friend =>
41 |       <div> {React.string(friend.firstName)} </div>
42 |     ),
43 |   )
44 | }

This has type:
  array(React.element(React.unkeyed))
But somewhere wanted:
  array(React.element(React.keyed))

The incompatible parts:
  React.unkeyed
  vs
  React.keyed
```

DOM Elements

- Right now there are two types of components - `<div />` and `<MyComponent />` are implemented in different ways
- One way to deal with this is to unify with a namespace like `React.DOM` and put each HTML element into this namespace

DOM Elements

- One big advantage here is that it means that props for each HTML element can be individually defined.

- Also opens up lowercase as a non-special case

```
[@react.component]
let myComp = ...;
let myElement = <myComp />;
```

DOM Elements - Result

```
React.DOM(
  <div>
    <button onClick={_event => dispatch(Click)}>
      {React.string(message)}
    </button>
    <div>
      {
        React.array(
          Belt.Array.map(friends, friend =>
            <div key=friend.id> {React.string(friend.firstName)} </div>
          ),
        )
      }
    </div>
  </div>
);
```

Others?

- Hooks
- Transitions/animations
- Nominal component types. Enforcing children.
- Other?

Thanks!

- https://github.com/rickyvetter/reason-conf-us
- https://github.com/reasonml/reason-react
- @rickyvetter