rivarjs

Reactive Instance Variable for JavaScript based on RxJS

`rivarjs` is a decentralized state management library that automates changes. The heart of `rivarjs` lies in an innovative `RIVar` datatype. `RIVar` stands for *Reactive Instance Variable*: a combination of *Reactive Variable* from FRP with *Instance Variable* (i.e., object's variable) from OOP.

Features

- **Extend-only assignments:** New assignments do not overwrite previous ones, but rather extend them.
- **Cyclical dependencies:** Variables can be declared in terms of each other, creating a dynamic and responsive system.
- **Automatic updates:** Changes to any variable automatically propagate to all dependent variables.

The internal implementation is that each variable is an *observable stream* from RxJS. Also the assigned expressions for these variables are implemented as observable streams. The observable stream of a variable is created from merging the observable streams of the whole assigned expressions.

Installation

To use `rivarjs`, you have two options. First, you can install it using `npm` by running the following command:

```
npm install rivarjs
```

Alternatively, for an `HTML` page, you need to include the `rivarjs` script and its required dependency, `RxJS`, by adding the following script tags:

```
<script src="https://unpkg.com/rxjs@^7/dist/bundles/rxjs.umd.min.js"></script>
<script src="https://unpkg.com/rivarjs/dist/rivar.umd.js"></script>
```

Once you have `rivarjs` available, you can import the necessary elements in your `JavaScript` code using the following syntax:

```
var { RIVar, lift, Signal } = rivarjs;
```

Usage

1. Variables

`var myRIVar=new RIVar();`

2. Lift

`var functionOverRIVars=lift((x, y) => x * y, firstRIVar, secondRIVar);`

3. Assignments

`myRIVar.set(functionOverRIVars);`

It is usually prefered to compose this with the previous step:
`myRIVar.set(lift((x, y) => x * y, firstRIVar, secondRIVar))`

Composition

```
class A {
   constructor() {
     this.firstRIVar = new RIVar();
     // you may assign this.firstRIVar
   }
 }

class B {
   constructor(a) {

     this.a = a;

     this.secondRIVar = new RIVar();
     this.thirdRIVar = new RIVar();

     this.a.firstRIVar.set(lift(mul, this.secondRIVar, this.thirdRIVar));

   }
 }
```

Inheritance

```
class A {
   constructor() {
     this.firstRIVar = new RIVar();
     // you may assign this.firstRIVar
   }
 }

class B extends A {

   constructor(a) {

     this.secondRIVar = new RIVar();
     this.thirdRIVar = new RIVar();

     this.FirstRIVar.set(lift(mul, this.secondRIVar, this.thirdRIVar));

   }
 }
```

Integration

React

`RIVarView` is a `React component` to render according to a `rivar`

```
import { RIVarView } from 'rivarjs/integration/react';
```

`RIVarView` takes `prop` `rivar` and `children prop` of a `render` function. The `render` function returns `JSX` of a `react` component according to `value` (at the time of rendering) and `change` (to transfer changes from events to the `rivar`).

```
<RIVarView rivar={rivar}>
  {({ value, change }) => {
    return <input
      type="number"
      value={value}
      onChange={(event) => change(event.target.value)}
    />;
  }}
</RIVarView>
```

Pure JavaScript

The following code initiates a connection between an instance of `RIVar` to an `HTML` element.

```
  function bind(inputID, variable) {

    var input = document.getElementById(inputID);

    input.addEventListener('input', (event) => {
      const value = event.target.value;
      variable.next(new Signal(value));
      input.style.fontStyle = "normal";
    });

    variable.subscribe((signal) => {
      if (input.value !== signal.value.toString()) {
        input.value = signal.value.toString();
        input.style.fontStyle = "italic";
      }
    });

  }
```