RealNode-API-for-reactive-programming

为响应式编程提供了若干实用的ES10类，基类为`RealNode`和`RealWorld`。

由于大多数的类有着极其复杂的属性和方法，此处先进行相关说明：

属性分级

类/实例的属性在使用频率和是否只读的方面被分为3级：

1. 常用属性
2. 慎用属性
3. 隐藏属性

级别越大，稳定性越差，越不建议使用！

未提及的属性一般被认为是隐藏属性。

方法分级

类/实例的方法在使用频率的方面被分为隐藏：

- 一 常用方法
- 二 慎用方法
- 三 隐藏方法

级别越大，稳定性越差，越不建议使用！

未提及的方法一般被认为是隐藏方法。

尽可能不要使用隐藏方法，因为这些方法针对性（功能性）极强，不适合也没必要在日常中使用。

各种类/对象/函数的使用指南

导入代码库

设计和使用思路

1. 事件循环类`RealWorld`
2. 响应式类`RealNode`
3. 对象响应式类`RealGroup`
4. 对象属性响应式类`RealTarget`

5. 元素属性响应式类`RealElement`
6. 自动流程对象`RealStory`
7. 异步流程对象`RealPromise`

   以下类和函数仅在浏览器环境中有效！！！

8. 异步画布元素类`RealCanvas`
9. 文件交互元素类`RealLoader`
10. select元素拓展类`RealSelect`
11. 组合元素类`RealComtag`
12. 块元素列表类`RealDivList`
13. 图片元素列表类`RealImgList`
14. 块元素排列类`RealDivQueue`
15. 函数：创建块元素选择类`createRealDivSelect()`
16. 函数：创建块元素搜索类`createRealDivSearch()`

导入代码库

- CJS标准：npm安装后使用`require`函数即可。
- ESM标准：找到代码库目录下的index.js文件，用文本编辑器打开，找到代码为`// export default`的那一行，取消其注释后即可用`import`关键字导入。

设计和使用思路

我们假设如下场景，并用代码逐个转译：

你是副部级干部沙瑞金。

```
var 我 = new RealNode({id: '沙瑞金',value: '副部'});
```

你是中管干部，所以如果没有中央批准的话，你就不能有任何调动。

```
var 中央批准 = new RealNode({id: '中央批准',value: false});        // 方便演示，简化判断流程
function 进行调动(value){
        if(中央批准.value) return this.protoSet(value);
        this.log('你无权对我进行调动。');
        return false;
}
我.set = 进行调动;
```

中央信任你的才干，交给你反腐的使命。你升任汉东省省委书记，级别为正部级。

```
中央批准.value = true;    // 中央批准通过
我.value = '正部';         // 升任正部级
中央批准.set(false);       // 重置中央批准
```

目前汉东省省委常委的结构大概是这样的。

```
var 汉东省省委常委 = {
        '省委书记': '沙瑞金',
        '省委副书记': '高育良',
        '京州市委书记': '李达康',
};
```

你很清楚，省委常委都是中管干部，你也无权对其进行调动。

```
汉东省省委常委 = new RealGroup({self: 汉东省省委常委, id: '汉东省省委常委'});
中央批准.value = true;    // 之前任命的，现在为了演示就暂时通过一下
汉东省省委常委.set({
        '省委书记': new RealNode({id: '省委书记',initValue: '沙瑞金', set: 进行调动}),
        '省委副书记': new RealNode({id: '省委副书记',initValue: '高育良', set: 进行调动}),
        '京州市委书记': new RealNode({id: '京州市委书记',initValue: '李达康', set: 进行调动}),
});
中央批准.set(false);       // 重置中央批准
```

你空降汉东省，使得国内外开始时刻注意汉东省省委常委的调动情况。

```
汉东省省委常委.addSetterListener(null,()=>console.log('汉东省省委常委发生了调动！！！'));
```

从北京调来的侯亮平同志查出了省委副书记高育良的违法乱纪事实，但无权进行查处。

```
汉东省省委常委.省委副书记 = '';      // 控制台输出日志 [object RealNode]{ 省委副书记 } : '你无权对我进行调动。
```

你认为省委副书记高育良的影响巨大，需要耐心处理，但没想到社会各界都十分关注高育良的动向。

```
汉东省省委常委.addSetterListener('省委副书记',()=>{
        var 汉东省省委常委现状 = 汉东省省委常委.get();        // 相当于{'省委书记': '沙瑞金','省委副书记': '
        var 汉东省省委常委现状 = 汉东省省委常委.value;        // 另一种等价的获取方法
        for(var 职务 in 汉东省省委常委现状) if(汉东省省委常委现状[职务] === '高育良') return console.log('
        console.log('高育良落马了！！！');
});
```

高育良也预料到有落马的一天。

```
汉东省省委常委.proxy();    // 返回
                                    // {
                                    //       '省委书记': [object RealNode]{ 省委书记 },
                                    //       '省委副书记': [object RealNode]{ 省委副书记 },
                                    //       '京州市委书记': [object RealNode]{ 京州市委书记 },
                                    // }
汉东省省委常委.proxy().省委副书记.react = function(){this.log('悔不当初啊。');};
```

你知道，一旦高育良被双规，汉东省官场将会发生巨大动荡。

```
汉东省省委常委.relate(我); // 一旦汉东省省委常委发生调动，我会立刻收到通知并进行通知转发
我.react = function(){
```

```
        var 汉东省省委常委现状 = 汉东省省委常委.get();         // 相当于{'省委书记': '沙瑞金','省委副书记': '
        var 汉东省省委常委现状 = 汉东省省委常委.value;         // 另一种等价的获取方法
        for(var 职务 in 汉东省省委常委现状) if(汉东省省委常委现状[职务] === '高育良') return this.log('高育
        this.log('汉东省官场不稳了。');
};      // 收到通知后，我会对形势变化做出反应判断
```

随着事态的不可控，中央批准双规高育良。

```
中央批准.value = true;    // 中央批准通过
汉东省省委常委.proxy.省委副书记 = '';        // 双规高育良，控制台输出日志 [object RealNode]{ 省委副书记 } :
                                                                    // 控制台输出日志 '高育良落马
// 汉东省省委常委.set({省委副书记: ''},true,true); //上一行代码的另一种写法
中央批准.set(false);      // 重置中央批准
        // 控制台输出日志 [object RealNode]{ 沙瑞金 } : '汉东省官场不稳了。'
```

RealWorld类

这是一个事件循环类，基于`setInterval()`函数实现。对该类的一个实例而言，每过一段固定时间将会调用实例慎用方法`_mainFn()`。

构造函数 `new RealWorld(timeSep, ...fnList)`

`timeSep`（可选）应为一个数值，否则默认为10，单位为毫秒。

`...fnList`（可选）应为`Function`类型，但不建议使用。

常用属性（只读）

- `timeSep` 实例属性，`Number`类型。

常用属性

- `paused` 实例属性。

  若为真值，则会暂停该实例的运行，否则恢复该实例的运行。

- `intervalFn` 实例属性，应为`Function`类型。

  会在实例慎用方法`_mainFn()`被调用时执行，若报错，则会被清除。

- `ifFn` 实例属性，应为`Function`类型。

  会在实例慎用方法`_mainFn()`被调用时执行，若报错，则会被清除。若执行的返回值是真值，则会被清除并尝试执行实例常用属性`soFn`。

- `soFn` 实例属性，应为`Function`类型。

  会在实例慎用方法`_mainFn()`被调用且实例常用属性`ifFn`被执行并返回真值时执行，执行后会被立即清除。

常用方法

- `destroy()` 实例方法，返回`undefined`。

  永久停止实例的运行，原理是使用`clearInterval()`函数。

- `setTimeSep()` 实例方法，返回`Boolean`类型。

  接收一个参数作为新的时间间隔，更改成功则返回`true`，否则反之。

- `then()` 实例方法，返回实例本身。

  接收一个参数`fn`，若`fn`是为`Function`类型，则插入到慎用实例属性（只读）`fnList`的第一位。

- `onceIf()` 静态方法，返回`Promise`类型。

  接收一个参数`ifFn`，必须为`Function`类型。当`ifFn`被执行并返回真值时，`onceIf()`方法返回的承诺将会被兑现。

慎用属性（只读）

- `fnList` 实例属性，`Array`类型。

  会在实例慎用方法`_mainFn()`调用时执行`fnList.pop()`并执行其返回值。

慎用属性

- `onload` 静态属性，应为`Promise`类型。

  浏览器环境下网页文档准备就绪时兑现。

慎用方法

- `_mainFn()` 实例方法，返回`undefined`。

  每过一段固定时间将会被调用。

RealNode类

这是一个响应式类，基于`Promise`类的微任务队列实现。对该类的一个实例而言，可以存储一个值，并在变更存储的值时会产生响应。

构造函数 `new RealNode(config, tryRealNode, ...relativeRNs)`

`config`（可选）应为一个对象，根据`config`的属性决定某些行为。

- `get`（可选）

  对实例常用属性`get`进行赋值。

- `set`（可选）

  对实例常用属性`set`进行赋值。

- `react`（可选）

  对实例常用属性`react`进行赋值。

- `id`（可选）

  初始化实例常用属性（只读）`id`时作为`description`。

- `info`（可选）

  对实例慎用属性`info`进行赋值。

- `value`（可选）

  对实例常用属性`value`进行赋值。

- `initValue`（可选）

  对实例常用属性`value`进行初始化。

`tryRealNode`（可选）

不建议使用。若为真值，当变更存储的值时将会尝试对新值中嵌套的`RealNode`实例进行解析。

`...relativeRNs`（可选）

应为`RealNode`类型或`Symbol`类型，但不建议使用。将会调用实例常用方法`relate()`，参数为`...relativeRNs`。

常用属性（只读）

- `id` 实例属性，`Symbol`类型。

  当实例的引用不小心丢失时，可以通过静态常用方法`search()`尝试找回。

常用属性

- `get` 实例属性，应为`Function`类型，且能够返回一个值。

- `set` 实例属性。

  读取值为实例慎用方法`realSet()`，写入值应为`Function`类型，应接收实例常用属性`get`的执行返回值并返回`Boolean`类型。所赋的值将在变更存储的值时被执行。

- `react` 实例属性，应为`Function`类型。

- `value` 实例属性。

  读取值为实例常用属性`get`的执行返回值，写入时将执行实例常用属性`set`，若返回真值，则会执行实例常用属性`react`和调用实例常用方法`notify()`。

- `display` 实例属性。

  读取值为`Boolean`类型，默认是`true`。若写入真值，将能够接收到其他实例的广播通知，否则反之且无法被静态常用方法`search()`查询。

常用方法

- `log()` 实例方法，返回`undefined`。

  接收若干参数`message`。源代码：`console.log(this+' :',...message)`。

- `notify()` 实例方法，返回`undefined`。

  将根据实例隐藏属性`relativeRNs`查询`RealNode`实例并依次生成微任务，将依次执行实例常用属性`react`和调用实例常用方法`notify()`。

- `relate()` 实例方法，返回`RealNode`实例或`undefined`。

  接收若干`RealNode`实例或`Symbol`类型作为参数，并尝试返回最后一个`RealNode`实例。

- `unrelate()` 实例方法，返回`Boolean`类型。

  接收若干`RealNode`实例或`Symbol`类型作为参数。

- `search()` 静态方法，返回`RealNode`实例或`undefined`。接收一个参数`id`，应为`Symbol`类型。

- `justNow()` 静态方法，返回`Promise`类型。

  接收一个参数`fn`，应为`Function`类型，在生成的一个微任务中执行后兑现返回值。

- `afterNow()` 静态方法，返回`Promise`类型。

  接收一个参数`fn`，应为`Function`类型，在生成的一个宏任务中执行后兑现返回值。

慎用属性

- `eventLoop` 静态属性，必须是`RealWorld`实例。

慎用方法

- `realSet()` 实例方法，返回`Boolean`类型。

  执行时接收四个参数`value`、`react`、`notify`、`noSelf`，将根据实例隐藏属性`relativeRNs`查询`RealNode`实例并依次生成微任务，将依次执行实例常用属性`react`和调用实例常用方法`notify()`。

- `time()` 静态方法，返回`Promise`类型。

  接收一个参数`promise`，若为`Function`类型则执行，若为`Promise`类型则等待兑现，最终返回值将兑现`{time: Number, value: any | Error}`。

隐藏属性

- `relativeRNs` 实例属性，必须是`Array`实例，且每个元素都必须为`Symbol`类型，即`RealNode`实例的常用实例属性（只读）`id`。

RealGroup类

继承RealNode类

这是针对对象的响应式类，是RealNode类的子类。对该类的一个实例而言，可以代理一个对象，并在代理变更对象的键值对时会产生响应。

构造函数 `new RealGroup({id, info, self})`

`self`（可选）默认是一个`null`为原型的空对象。必须是一个对象，否则会报错！（注意：根据相同对象创建的`RealGroup`实例是同一个实例！）

`id`（可选）初始化常用实例属性（只读）`id`时作为`description`。

`info`（可选）对实例慎用属性`info`进行赋值。

常用属性（只读）

- `proxy` 实例属性，`Proxy`类型。

  对该属性的读写操作将完全转移到构造实例时的参数`self`对象上。

- `get` 实例属性，返回实例隐藏方法`protoGet()`。

  执行时：若没有参数，则返回构造实例时的参数`self`对象的浅拷贝；接收一个参数`keyOrkeyObj`，若是一个对象，则返回一个`null`为原型的相同结构的对象，否则返回对应键的值。

- `set` 实例属性，返回实例隐藏方法`realSet()`。

  执行时接收一个参数`value`，`value`必须是对象，不能读取其原型链上的属性。

- `react` 实例属性，返回实例隐藏方法`protoReact()`。

常用方法

- `keys()` 实例方法，返回`Array`实例，每个元素为`String`类型或`Symbol`类型。

  接收一个参数`all`，若为真值，则返回值包括`Symbol`类型和不可枚举的键。

- `addSetterListener()` 实例方法，返回`undefined`。

  接收两个参数`ifKeyOrFn`和`listener`，`ifKeyOrFn`必须为`String`类型或返回`Boolean`类型的`Function`类型，`listener`必须为`Function`类型。

慎用属性（只读）

- `listenerMap` 实例属性，`Map`类型。

  键为`String`类型或`Function`类型，值为`Array`类型，所有元素为`Function`类型。

- `getByFilter()` 实例方法，返回一个`null`为原型的含对应键值对的对象。

  接收一个参数`filterFn`，必须为`Function`类型，根据筛选出的键返回一个`null`为原型的含对应键值对的对象。

隐藏方法

- `proxy()` 实例方法，返回构造实例时的参数`self`对象。

  此方法仅在找回原对象引用时使用。

- `createDeepGroup()` 静态方法，返回`RealGroup`类型。

  接收一个参数`obj`，最好是对象。

RealTarget类

继承RealNode类

这是一个键值单向绑定的响应式类，用于尽可能减少对某个对象的单个属性的赋值操作以提高性能。对该类的一个实例而言，可以存储一个值、绑定一个属性名和一个固定对象，当变更存储的值之后，会执行实例常用属性`transform`（`Function`类型）将存储的值经过函数变换后赋值给绑定的对象的相应属性并产生响应。

从语义上说，`RealTarget`类主要针对对象，`RealElement`类主要针对`HTML`元素。然而实际上，两者在实例属性和实例方法上没有任何区别，仅在静态属性和静态方法上有明显差异。

在实际开发中，`RealTarget`类常用于非浏览器环境中。

构造函数 `new RealTarget({self, key, transform}, config, tryRealNode, ...relativeRNs)`

`{self, key, transform}`必须提供一个对象用于解构参数。这个对象的属性必须满足以下要求。

- `self`

  必须是需要绑定的对象，否则直接报错。

- `key`

  应是需要绑定的属性名。

- `transform`（可选）

  应为`Function`类型。用于初始化实例常用属性`transform`。

`config`（可选）应为一个对象，根据`config`的属性决定某些行为。

- `get`（可选）

  对实例常用属性`get`进行赋值。

- `set`（可选）

  对实例常用属性`set`进行赋值。

- `react`（可选）

  对实例常用属性`react`进行赋值。

- `id`（可选）

  初始化常用实例属性（只读）`id`时作为`description`。

- `info`（可选）

  对实例慎用属性`info`进行赋值。

- `value`（可选）

  对实例常用属性`value`进行赋值。

`tryRealNode`（可选）

不建议使用。若为真值，当变更存储的值时将会尝试对新值中嵌套的`RealNode`实例进行解析。

`...relativeRNs`（可选）

应为`RealNode`类型或`Symbol`类型，但不建议使用。将会调用实例常用方法`relate()`，参数为`...relativeRNs`。

常用属性（只读）

- `isElement` 实例属性，`Boolean`类型。

  判断实例常用属性`self`是否为`HTML`元素。

常用属性

- `self` 实例属性，对象类型。

  当对其进行赋值时，如果赋的值不是对象，会直接报错。

- `key` 实例属性，任何类型。

  建议只赋值为`String`类型，`Number`类型或`Symbol`类型。

- `transform` 实例属性，`Function`类型。

  应接收一个参数并返回一个值。

常用方法

- `fix()` 实例方法，返回实例自身。

  调用该方法时，会接收存储的值以执行实例常用属性`transform`（`Function`类型）进行函数变换并将返回值赋值到绑定对象的对应属性上，是一个同步过程。

- `clearClassName()` 实例方法，返回`Boolean`类型。

  如果绑定的对象是一个`HTML`元素，则会清空其类名并返回`true`，否则反之。

- `addClassName()` 实例方法，返回`Boolean`类型。

如果绑定的对象是一个HTML元素，则会执行该HTML元素的classList的add()方法且接收所有参数并返回true，否则反之。

- toggleClassName() 实例方法，返回Boolean类型。

  应接收一个参数className，className应为String类型。如果绑定的对象是一个HTML元素，则会执行该HTML元素的classList的toggle()方法且接收参数className，执行之后其返回值将被返回（Boolean类型），否则返回false。

- removeClassName() 实例方法，返回Boolean类型。

  如果绑定的对象是一个HTML元素，则会执行该HTML元素的classList的remove()方法且接收所有参数并返回true，否则反之。

- getIndexWithin() 实例方法，返回Number类型。

  返回值是绑定的HTML元素在其父元素中的HTML元素排列索引。原理是通过绑定的对象的previousElementSibling属性的真假值判断，所以正常情况下，即使绑定的对象不是HTML元素也不会报错。

- removeClassName() 实例方法，返回Boolean类型。

  如果绑定的对象是一个HTML元素，则会执行该HTML元素的classList属性的remove()方法且接收所有参数并返回true，否则反之。

- applyCSS() 实例方法，返回Boolean类型。

  如果绑定的对象不是HTML元素，则会直接报错。

  接收两个参数selfSelector、classNameOrRuleObjObj，其中参数selfSelector必须为String类型，参数classNameOrRuleObjObj应为一个类名（String类型）或一个对象。

  selfSelector和classNameOrRuleObjObj（对象类型）的属性名都会被视作CSS选择器，按照一定规律进行【绑定的HTML元素的id属性 + 参数selfSelector + 参数classNameOrRuleObjObj的属性名】的字符串拼接，如果绑定的HTML元素没有id属性，则会自动进行随机id属性注册。classNameOrRuleObjObj（对象类型）的每一个属性都必须是符合CSS标准的字符串键值对对象。如果classNameOrRuleObjObj为String类型，则会对当前注册过的类名进行检索并自动获取所需对象。

  底层原理是CSSStyleSheet.insertRule()方法。

- clone() 实例方法，返回RealTarget类型。

  接收三个参数keepValue（可选）、fix（可选）、deepCopyRelativeRNs（可选），其中所有参数应为Boolean类型。

  返回的RealTarget对象拥有相同的绑定对象、绑定属性名和相同引用的实例隐藏属性relativeRNs，参数keepValue决定是否拥有相同的存储值，参数fix决定在返回时是否调用实例常用方法fix()，参数deepCopyRelativeRNs决定是否对实例隐藏属性relativeRNs进行深复制。

- searchByObj() 静态方法，返回包含RealTarget对象的Array对象。

接收一个参数`element`，应为对象，搜索并返回所有绑定对象为参数`element`的`RealTarget`对象组成的`Array`对象。

RealElement类

继承RealTarget类 继承RealNode类

这是一个键值单向绑定的响应式类，用于尽可能减少对某个`HTML`元素的单个属性的赋值操作以提高性能。

从语义上说，`RealTarget`类主要针对对象，`RealElement`类主要针对`HTML`元素。然而实际上，两者在实例属性和实例方法上没有任何区别，仅在静态属性和静态方法上有明显差异。

在实际开发中，`RealElement`类常用于浏览器环境中。

构 造 函 数 `new RealElement({self, key, transform}, config, tryRealNode, ...relativeRNs)`

详见RealTarget构造函数

常用方法

- `createImg()` 静态方法，返回`RealElement`类型。

  等价于执行并返回`new RealElement({self: document.createElement('img'), key: 'src'})`。

- `createVideo()` 静态方法，返回`RealElement`类型。

  等价于执行并返回`new RealElement({self: document.createElement('video'), key: 'src'})`。

- `createAudio()` 静态方法，返回`RealElement`类型。

  等价于执行并返回`new RealElement({self: document.createElement('audio'), key: 'src'})`。

- `createDiv()` 静态方法，返回`RealElement`类型。

  接收两个参数`id`和`initValue`。等价于执行并返回`new RealElement({self: document.createElement('div'), key: 'textContent'}, {id, initValue})`。

- `createTextarea()` 静态方法，返回`RealElement`类型。

  等价于执行并返回`new RealElement({self: document.createElement('img'), key: 'src'})`。

- `makeElement()` 静态方法，返回一个`HTML`元素。

  接收三个参数`tagName`、`config`、`cssConfig`，其中参数`tagName`应是一个`HTML`标签字符串，或一个`HTML`元素（即为返回的`HTML`元素），参数`config`应是描述返回的`HTML`元素的属性的键值对对象，参数`config`应是描述返回的`HTML`元素的CSS样式属性的键值对对象。

- `getDomByString()` 静态方法，返回一个`HTML`元素或`null`。

  接收一个参数`innerHTML`，应为`String`类型，返回第一个被解析出来的`HTML`元素。

- `addCSSRules()` 静态方法，返回该静态方法自身。

  接收两个参数`prefix`和`ruleObjObj`，其中参数`prefix`必须为`String`类型或`String`实例组成的`Array`类型，参数`ruleObjObj`应为一个对象。

  `prefix`（或`prefix`数组的元素）和`ruleObjObj`（对象类型）的属性名的拼接都会被视作CSS选择器，按照一定规律进行【参数`prefix` + 参数`classNameOrRuleObjObj`的属性名】的字符串拼接，`ruleObjObj`（对象类型）的每一个属性都必须是符合CSS标准的字符串键值对对象。如果`prefix`正好是一个类名的选择器`.${className}`，那么将会对该类名及其对应对象进行注册。

  底层原理是`CSSStyleSheet.insertRule()`方法。

慎用属性

- `keyboardController` 静态属性，应是包含`previous`、`next`、`enter`、`back`等4个属性的对象。

  该对象的这4个属性应该是键盘按键的字符串，其用途详见静态隐藏方法`applyKeyboardController()`。

慎用方法

- `addEventListenerBySelectors()` 静态方法，返回`undefined`。

  接收三个参数`selectors`、`type`、`listener`，其中`selectors`，相当于`document.querySelectorAll( selectors ).forEach( element => element.addEventListener(type, listener) )`，但是有两个特点：

  1. 所有事件都委托到`document`对象，所以不能侦听到没有冒泡的事件，例如正常的`'change'`事件；
  2. 对于一个元素，只要能够被选择器`selectors`选择到，就有可能触发侦听器`listener`。

- `defaultInit()` 静态方法，返回`Promise`类型。

  能够添加一系列样式，之后生成新的`Promise`实例，覆盖`RealWorld`类的静态常用属性`onload`并返回之。

隐藏方法

- `applyKeyboardController()` 静态方法，返回`undefined`。

  接收若干参数，都应为`HTML`元素或`RealElement`类型。将会对`HTML`元素或`RealElement`实例的常用属性`self`添加一个类名`'keyboardController'`。

  一个`'keydown'`类型的事件侦听器已被设置，它的作用是：如果按下的按键的字符串与静态慎用属性`keyboardController`的四个属性之一完全相同，那么将会执行对应的以下操作：

  - `previous`属性：选中当前元素在其父元素的子元素节点中的前一个元素节点。
  - `next`属性：选中当前元素在其父元素的子元素节点中的前一个元素节点。
  - `enter`属性：选中当前元素的后代元素节点中的第一个含类名`'keyboardController'`的元素，如果不存在，则触发当前元素的`'click'`事件。
  - `back`属性：选中当前元素的前代元素节点中的第一个含类名`'keyboardController'`的元素，如果不存在，则选中当前元素。

- `cancelKeyboardController()` 静态方法，返回`undefined`。

  接收若干参数，都应为`HTML`元素或`RealElement`类型。将会对`HTML`元素或`RealElement`实例的常用属性`self`移除一个类名`'keyboardController'`。

RealStory对象

这是一个自动运行的流程编辑器对象。`RealStory`对象对于你而言，你可以调用`then()`方法将一个函数设置为宏任务中的一个微任务，也可以调用`newPage()`方法创建`RealStory`对象的一个子级流程编辑器对象，其类型与`RealStory`对象是同一个类型，这意味着你也能够在该子级流程编辑器对象调用`then()`方法，甚至可以继续调用`newPage()`方法创建孙级流程编辑器对象。

假设`RealStory`对象是一个称“方明正”的人，其长子称“方明正之子 - 0”，“方明正之子 - 0”的长子称“方明正之孙 - 0 - 0”，每个人的心愿即为调用`then()`方法设置的微任务。此时参考以下结构：

我辈	子辈	孙辈
“方明正”	“方明正之子 - 0”	“方明正之孙 - 0 - 0”
	-	“方明正之孙 - 0 - 1”
-	“方明正之子 - 1”	“方明正之孙 - 1 - 0”
	-	“方明正之孙 - 1 - 1”

那么这7个人的心愿的满足顺序为：

1. “方明正”
2. “方明正之子 - 0”
3. “方明正之孙 - 0 - 0”
4. “方明正之孙 - 0 - 1”
5. “方明正之子 - 1”
6. “方明正之孙 - 1 - 0”
7. “方明正之孙 - 1 - 1”

常用属性（只读）

- `index` 属性，`Number`类型。

  这是当前流程编辑器对象在其父级流程编辑器对象的子级中的次序，以`0`为第一位，若当前流程编辑器对象为`RealStory`对象，则次序为`-1`。

- `StoryPromise` 属性，匿名类。

  这是一个匿名类，进行构造或直接调用都会返回一个对象。该类等价于`Promise.withResolvers()`方法。

常用方法

- `newPage()` 方法，返回流程编辑器对象。

  返回的是当前流程编辑器对象的一个新的子级流程编辑器对象。

- `then()` 方法，返回当前流程编辑器对象自身。

  接收一个参数`fn`，应是`Function`类型。参数`fn`会被追加到当前流程编辑器对象的隐藏属性`fnList`中。

- `getNextPage()` 方法，返回流程编辑器对象或`undefined`。

  关于返回的流程编辑器对象，其父级与当前流程编辑器对象的父级相同，但次序靠后一位。

- `getPreviousPage()` 方法，返回流程编辑器对象或`undefined`。

  关于返回的流程编辑器对象，其父级与当前流程编辑器对象的父级相同，但次序靠前一位。

- `newPromiseObj()` 方法，返回常用属性（只读）`StoryPromise`构造的新实例。

  源代码：

  ```
  const temp = new StoryPromise;
  return this.then(()=>temp.promise),temp;
  ```

- `launch()` 方法，返回`Promise`类型。

  这是异步方法。在调用该方法到返回值兑现期间，特定函数将按一定规律被执行。（其运作机制详见`RealStory`对象介绍）

  `RealStory`对象不需要手动调用`launch()`方法，因为其内部已经设置了定时器自动触发，所以虽然`launch()`方法是基于`Promise`类的微任务队列实现，但是其执行时机却是在宏任务时间。

慎用属性

- `info` 属性，类型不限。

  看似没用，其实还是有一点点用的，吧。

慎用方法

- `newPrivatePage()` 方法，返回`Promise`类型。

  接收一个参数`fn`，必须是`Function`类型。参数`fn`能够接收当前流程编辑器对象的一个新的子级流程编辑器对象作为其第一个参数。

隐藏属性

- `ofStory` 属性，应是流程编辑器对象。

  一般是当前流程编辑器对象的父级流程编辑器对象。

- `pages` 属性，应是包含流程编辑器对象的`Array`实例。

  一般是当前流程编辑器对象的子级流程编辑器对象组成的`Array`实例。

- `fnList` 属性，应是包含函数的`Array`实例。

  该属性中的每个元素应该都是通过调用当前流程编辑器对象的常用方法`then()`添加的函数。

RealPromise对象

这是一个异步流程对象，基于`Promise`类的微任务队列实现，但原型链上并没有`Promise`类的原型。

你可以把`RealPromise`对象当作一个`Promise`实例一样调用`then()`、`catch()`、`finally()`等方法，但不同之处在于：

1. 这些方法的返回值都是`RealPromise`对象自身；
2. `then()`方法会把兑现值追加记录到常用属性`list`（`Array`类型）之中。

常用属性（只读）

- `length` 属性，`Number`类型。

  获取当前常用属性`list`的长度。

常用属性

- `list` 属性，`Array`类型。

  里面记录了`RealPromise`对象当前已执行了的微任务的兑现值。

- `self` 属性，`Promise`类型。

  这是当前微任务。

常用方法

- `newOne()` 方法，返回类型与`RealPromise`对象相同的新对象实例。

  如果你需要更多的`RealPromise`对象，可以使用这个方法。

- `catch()` 方法，返回`RealPromise`对象自身。

  接收一个参数`onrejected`，应为`Function`类型。等价于在返回前执行代码`this.self = this.self.catch(onrejected)`。

- `finally()` 方法，返回`RealPromise`对象自身。

  接收一个参数`onfinally`，应为`Function`类型。等价于在返回前执行代码`this.self = this.self.finally(onfinally)`。

- `then()` 方法，返回`RealPromise`对象自身。

  接收两个参数`onfulfilled`和`onrejected`，都应为`Function`类型。等价于在返回前执行代码`this.self = this.self.then(onfulfilled, onrejected)`。

- `tryHandler()` 方法，返回`Promise`类型。

  这是异步方法，接收接收两个参数`handler`和`onerror`（可选），都应为`Function`类型。

  调用该方法时，该方法会等待当前的常用属性`self`兑现或拒绝，之后依次倒序将常用属性`list`中的元素作为参数传入`handler`（`Function`类型）尝试执行，一旦成功，该方法的返回值就会以`handler`（`Function`类型）的返回值兑现。

- `require()` 方法，返回`Promise`类型。

  接收一个参数`path`，应为`String`类型。

  - 如果存在NodeJS环境，则返回值等价于代码`import(path).then(temp=>temp.default)`；
  - 如果存在浏览器环境，则采用`<script:src=path>`标签将参数`path`视为JS代码文件路径动态执行，如果执行成功，返回值就会在代码加载完后兑现；
  - 如果以上两种环境均不存在，返回值将会以一个错误拒绝。

慎用方法

- `_push()` 方法，返回接收的第一个参数。

  等价于执行代码`return v === this.list[this.list.length - 1] || this.list.push(v),v`。

以下类和函数仅在浏览器环境中有效！！！

以下类和函数仅在浏览器环境中有效！！！

以下类和函数仅在浏览器环境中有效！！！

RealCanvas类

继承`RealElement`类 继承`RealTarget`类 继承`RealNode`类

这是

构造函数 `new RealCanvas(id, width, height, tryRealNode, ...relativeRNs)`

常用属性（只读）

- `ctx` 实例属性
- `img` 实例属性
- `imgW` 实例属性
- `imgH` 实例属性

常用属性

- `width` 实例属性
- `height` 实例属性
- `clearBeforeDraw` 实例属性
- `temp` 实例属性
- `opacity` 实例属性
- `noCache` 静态属性

慎用属性

- `loaded` 实例属性

常用方法

- `clearAsync()` 实例方法
- `testSrc()` 实例方法
- `clear()` 实例方法
- `resizeBySrc()` 实例方法
- `strN()` 静态方法
- `toBlob()` 实例方法
- `getImageBySrc()` 静态方法

慎用方法

- `multiDrawSrcArray()` 实例方法
- `clearShape()` 实例方法
- `animate()` 实例方法

隐藏方法

- `applyMouseClear()` 实例方法

RealLoader类

继承RealElement类 继承RealTarget类 继承RealNode类

这是

构 造 函 数 `new RealLoader(isDownload, fileName, dataGetter, {innerHTML, onerror, onloadend})`

常用属性（只读）

- `type` 实例属性
- `files` 实例属性

常用属性

- `onerror` 实例属性
- `onloadend` 实例属性
- `fileName` 实例属性
- `dataGetter` 实例属性

隐藏属性

- `fs` 静态属性

常用方法

- `load()` 实例方法
- `getArrayBufferFrom()` 静态方法
- `load()` 静态方法

RealSelect类

继承`RealElement`类 继承`RealTarget`类 继承`RealNode`类

这是

构 造 函 数 `new RealSelect(id, multiple, optionConfig, tryRealNode, defaultKey, defaultValue, onchange)`

常用属性（只读）

- `list` 实例属性

常用属性

- `defaultKey` 实例属性
- `defaultValue` 实例属性

常用方法

- `getValueIndexs()` 实例方法

RealComtag类

继承RealElement类 继承RealTarget类 继承RealNode类

这是

构造函数 `new RealComtag(id, tryHTML, optionList, tryRealNode, selfAssign)`

慎用属性

- `comtagClassMap` 静态属性

慎用方法

- `defineComtagClass()` 静态方法
- `createByClassName()` 静态方法

RealDivList类

继承RealElement类 继承RealTarget类 继承RealNode类

这是

构造函数

常用属性（只读）

- `childrenList` 实例属性

慎用属性

- `divListClassMap` 静态属性

常用方法

- `getRealEmentList()` 实例方法
- `createList()` 静态方法

慎用方法

- `getIdDict()` 实例方法
- `defineDivListClass()` 静态方法

- `createByClassName()` 静态方法

RealImgList类

继承`RealDivList`类 继承`RealElement`类 继承`RealTarget`类 继承`RealNode`类

这是

构造函数

常用方法

- `cloneImgList()` 实例方法

RealDivQueue类

继承`RealDivList`类 继承`RealElement`类 继承`RealTarget`类 继承`RealNode`类

这是

构造函数

常用属性（只读）

- `queueArray` 实例属性

常用方法

- `getListQueue()` 实例方法

隐藏方法

- `applyQueue()` 实例方法

createRealDivSelect()函数

这是

``

createRealDivSearch()函数

这是

``

敬请期待后续更新