Expression String

> 单纯的值字面量还无法解决应对各种各样多变的业务逻辑. 所以我们必须要一种可以动态计算的方式来对数据进行的处理.

初识

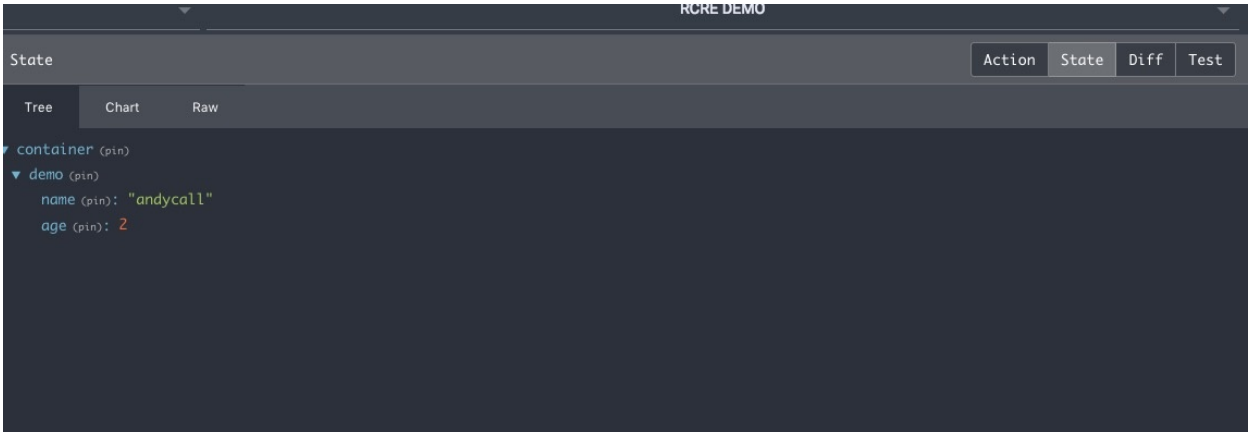
1 + 1 == 2

我们可以来一个非常简单的数学运行运算来开始本章的介绍.

```
{
    "type": "container",
    "model": "demo",
    "data": {
        "name": "andycall",
        "age": "#ES{1 + 1}"
    },
    "children": [
        {
            "type": "text",
            "text": "helloworld"
        },
        {
            "type": "text",
            "text": "another text"
        }
    ]
}
```

这个例子里面, `data`属性使用了一个Expression String字符串. 它运行位于`#ES{}`内部的`1 + 1`. 并计算出最终结果为2.

我们可以在Redux DevTools中看到最终运行的结果



语法

一个Expression String实际上是一个可执行的javaScript运行环境, 在这个环境内, 可以执行任何符合javaScript语法的代码. 例如字符串, 对象, 函数, 数组, 以及javaScript内置了Object, Array等函数. 不过对安全性的考虑, 这个环境内无法对DOM方法和window方法进行调用.

每一个Expression String都是以`#`字符串为开头. 之后跟上`ES`这2个字母. 用于表述这个是一个Expression String字符串, 之后跟上一个闭合的大括号, 可以动态执行的代码就需要放在大括号内部.

这里有几个运行的例子:

基础运算

```
#ES{1 + 1}  ==> 2
#ES{1 + 2 + 3} ==> 5
#ES{'1' + 1}  ==> '11' JS独特的特性
#ES{1} + #ES{2} ==> 3
#ES{1} + #ES{'2'} ==> '12'
```

从上个这个例子可见, Expression String不光能单个运行, 还能够多个进行组合. 执行过程中内部实际上是使用javaScript引擎来执行, 所以典型的javaScript类型问题, 在Expression String也会有一样的效果.

字符串模板

Expression String 一样也能当作是字符串模板来使用. 这在使用变量来拼接一个字符串是非常有效的方式

```
you are the #ES{1 + 1}th ==> you are the 2th
```

解析器只会处理在`#ES`内部的代码. 其他的字符串都只会当作是普通字符串进行处理

内嵌函数

Expression String还可以内嵌匿名函数进行运算, 这在一些特殊的场景下是非常有效的.

```
#ES{1 + (function(){function add(a, b) {return a + b;}return add(1, 2) + add(3, 4)})
()} ==> 11
```

这个例子在Expression String内部嵌入了一个自运行匿名函数, 匿名函数内部再调用闭包内部的1个函数. 最终执行得的最终结果为`11`

在RCRE的`filter`函数支持之前, 对于特殊的数据处理方案, 可以采用内嵌匿名函数的方式来进行一些特殊的数据处理.

对象和内置函数

Expression String一样可以操作javaScript对象.

```
#ES{Object.keys({name: 1, age: 2})} ==> ['name', 'age']
#ES{#ES{{arr:[{name: 1},{name: 2}]}["arr"].length}} ==> 2
```

RCRE内置变量

RCRE提供了一些内置的变量, 来方便开发者在Expression String中对当前`container`组件的数据模型进行操作.

$data

`$data`变量提供了在Expression String获取当前数据模型数据的功能. `$data`是一个普通的javaScript对象. 其中的各个属性可以直接在Redux Devtools进行查看. 例如:

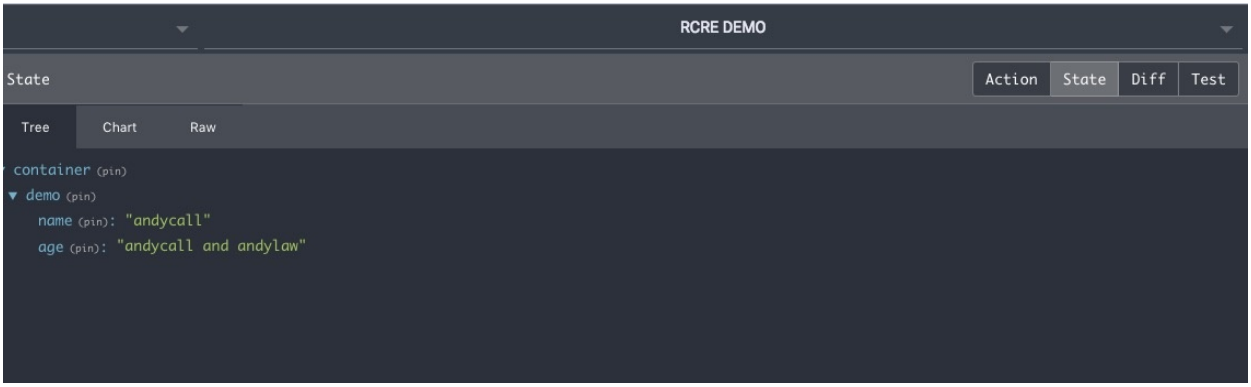
```
{
    "type": "container",
    "model": "demo",
    "data": {
        "name": "andycall",
        "age": "#ES{$data.name} + ' and andylaw'"
    },
    "children": [
        {
            "type": "text",
            "text": "helloworld"
        },
        {
            "type": "text",
            "text": "another text"
        }
    ]
}
```

这个例子中, `container`组件会在初始化时候, 对`data`属性中的字段进行Expression String解析. 并在执行环境中嵌入`$data`属性. 执行过程中`$data`中是一个含有非Expression String的值的常规对象. 所以执行`#ES{$data.name} + ' and andylaw'`的时候, `$data`的值为

```
{"name": "andycall"}
```

> 而含有`#ES{}`这样的字符串会在初始化的时候临时忽略掉, 所以目前功能还不支持`$data`递归调用自身.

这里例子最终运行的结果在Redux devTools中看到的应该是这个样子:


RCRE DEMO
State
Action
State
Diff
Test
Tree
Chart
Raw
container (pin)
demo (pin)
name (pin): "andycall"
age (pin): "andycall and andylaw"