嵌套的Container组件

> container组件能够为字级组件传递数据, 同样, container也可以给container传递数据

结合上面的几个章节, 我们已经清楚. 一个`container`组件的子级组件, 都可以获取到来自父级的`$data`属性.

可是, 如果遇到子级的组件也是一个`container`组件的时候, 这个情况就需要专门来解释了.

数据模型的优先级

多层的`container`组件的出现, 必然会牵扯出优先级顺序的问题. 数据模型的优先级决定着当父级的`container`和字级的`container`中含有相同属性值冲突的时候, 相互之间merge的顺序.

父级为先

在RCRE中, 最顶层的`container`组件的数据模型拥有最高的优先级. 也就是说, 如果不显示声明的话, 默认情况下, 最顶层的`container`组件的属性, 会自动同步到字级的任意一层`container`组件的数据模型中.

例如:

```
{
    "body": [
        {
            "type": "container",
            "model": "textField",
            "data": {
                "name": 1
            },
            "children": [
                {
                    "type": "container",
                    "model": "innerTextField",
                    "data": {
                        "name": 2
                    },
                    "children": [
                        {
                            "type": "text",
                            "text": "#ES{$data.name}"
                        },
                        {
                            "type": "container",
                            "model": "deepinnerText",
                            "data": {
                                "name": 3
                            },
                            "children": [
                                {
                                    "type": "text",
                                    "text": "#ES{$data.name}"
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ]
}
```

这个例子展示了连续3层`container`嵌套的场景. 最顶层的`container`组件, 中间的`container`组件和最底层的`container`组件.

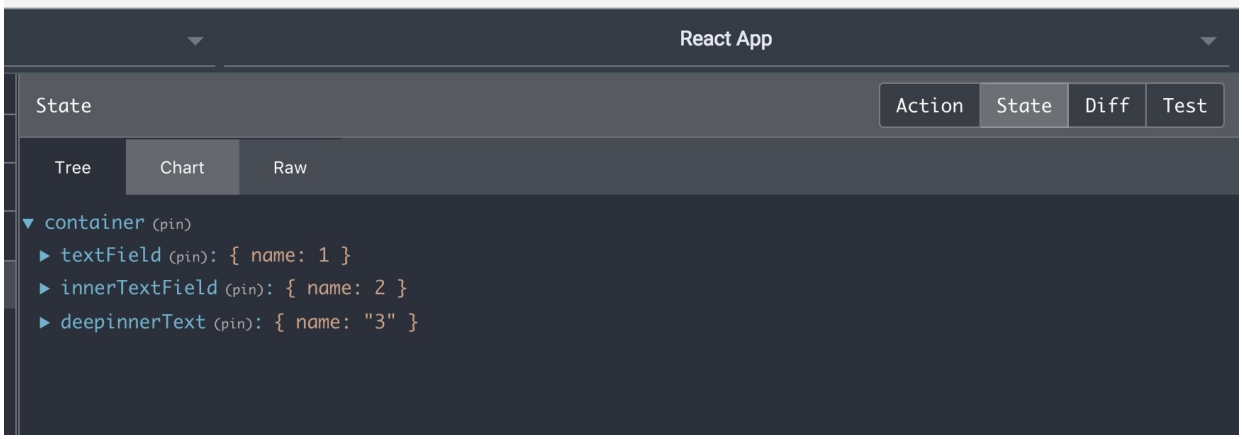
这个例子的运行结果是

1

1

可见, 当最顶层的`container`含有`name` = 1这个属性的时候, 即使字级的组件也含有`name`属性, 也不会有任何用处. 在RCRE渲染的时候, 会忽略来自原先数据模型中的值, 而采用父级的属性.

不过有个地方值得注意, 虽然页面显示的是2个1, 但是在Redux DevTools的查看时, 数据模型还是按照JSON配置的样子.



因为真正决定页面的内容是RCRE的`container`组件本身, 而不是数据模型的值. 而属性值的merge操作, 是在从数据模型渲染到页面的阶段时完成的.

优先级显性声明

字级组件如果没有显性生命的话, 那么来自父级的属性将会无法阻挡. 这是非常不利于开发的.

RCRE提供了一个内置属性 —— **parentMapping**, 这是一个常规对象, 对象的每一个键值都是Expression String. 通过**parentMapping**, 用户能自行决定来自父级的数据模型和子级的数据模型改如何进行merge.

parentMapping每个Expression String的运行上下文中将嵌入2个内置对象:

- $data 当前组件的数据模型
- $parent 父级组件的数据模型

这样用户可以自行实现逻辑来进行属性merge.

例如:

```
{
    "body": [
        {
            "type": "container",
            "model": "textField",
            "data": {
                "name": 1
            },
            "children": [
                {
                    "type": "container",
                    "model": "innerTextField",
                    "data": {
                        "name": 2
                    },
                    "parentMapping": {
                        "name": "#ES{$data.name}"
                    },
                    "children": [
                        {
                            "type": "text",
                            "text": "#ES{$data.name}"
                        },
                        {
                            "type": "container",
                            "model": "deepinnerText",
                            "data": {
                                "name": 3
                            },
                            "parentMapping": {
                                "name": "#ES{$data.name + $parent.name}"
                            },
                            "children": [
                                {
                                    "type": "text",
                                    "text": "#ES{$data.name}"
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ]
}
```

外层组件向所有的子级组件传递`{name : 1}`, 不过第二层的`container`组件的parentMapping显示它只需要资深的`name`属性, 所以它的Text组件显示是2, 最底层的parentMapping显示它将父级的`name`属性值和自己的`name`属性值进行相加. 所以它的Text组件显示是5.

这个例子的运行结果是