**redux-form V.7.4.2学习笔记--Field解析（完）**

2018年07月27日 15:50:18 [weixin\_34209406](https://me.csdn.net/weixin_34209406) 阅读数 13

版权声明：本文为博主原创文章，遵循[CC 4.0 by-sa](http://creativecommons.org/licenses/by-sa/4.0/)版权协议，转载请附上原文出处链接和本声明。

本文链接：<https://blog.csdn.net/weixin_34209406/article/details/85176782>

**说明**

原本我想把整个redux-form中知识点与使用技巧通过这个小系列短文全面总结出来的，但是此过程中发现问题的确不少。但同时，在参考国内外一些相关资源的同时，又发现了一个比redux-form更值得研究的东西。先卖个关子（可能有少数朋友已经有所了解），稍过一些时间我会专门撰文介绍。因此，临时把本文作为此小系列的结束篇，敬请有志于学习redux-form有朋友原谅（当然，在你了解到我介绍的那个更好用的form wrapper后你更为有所体谅）。

**引言**

redux-form官方网站提供了操作form的许多API，其中最重要的无外乎三个：reduxForm(config:Object) 、props和 <Field/>。

本文专注于分析<Field/>的基本使用方法及注意事项，但是阅读本语言的前提是需要你先初步掌握reduxForm(config:Object) 和props这两个API的使用。有关这两个API，在前面的几篇中我们已经探讨了许多，其实还有很多更深入的内容会在接下来的文章中进行解析。使用一个单篇来介绍<Field/>这个API的另一个原因是，这个API本身比较复杂与典型，掌握了这个API使用后，你会轻松把握另外两个相关联的API：<Fields/>和<FieldArray/>。

**<Field/>的作用**

<Field/>远非官方基本示例中使用得那么简单。其实，所有需要与Redux store 数据连接的表单中的输入组件（包括可能的多层次嵌套输入组件），都可以用 <Field/>实现。

**<Field/>使用三原则**

在正确使用<Field/>这个API之前，你需要牢记下面三条基本原则：

**（一）必须提供name属性**

此属性值可以是简单的字符串，如 userName、password，也可以是使用点和中括号串联表达的代表路径含义的复杂的字符串结构，如 contact.billing.address[2].phones[1].areaCode。

**（二）必须提供component属性**

此属性值可以是一个React组件（Component）、无状态函数（stateless function）或者是DOM所支持的默认的标签（分别是input、textarea和select)元素。

**（三）其他属性**

能够传递给<Field/>的属性有许多。如前方所强调的，其中name和component属性是必须提供的，截止目前最新版本为止，<Field/>还有下面一些属性是可选提供的：

* format
* normalize
* onBlur
* onChange
* onDragStart
* onDrop
* onFocus
* **props**
* parse
* **validate**
* **warn**
* withRef
* immutableProps

官文原文中说法是“All other props will be passed along to the element generated by the component prop.”直译过来，好像是“其他所有属性会被传递给由component属性生成的元素”。但是，这种理解本人感觉十分不妥，又有同学[Ted Yuen](https://github.com/tedyuen/react-redux-form-v6-example/blob/master/README.md)的翻译成“其他所有属性会通过prop传递到元素生成器中。如 className。”感觉也不是很恰当。还是让我们随着本文最后面复杂的示例讨论来确定如何理解这一句吧。

**使用<Field/>的前提——导入API**

var Field = require('redux-form').Field; // ES5

import { Field } from 'redux-form'; // ES6

**component属性使用三情形**

<Field/>组件的属性中，component这个属性相对比较复杂，也非常重要，下面作专门讨论。从源码分析来看，component属性最终是被传递给React.createElement()的，于是component存在三种类型的取值，如下所示。

**使用方式一：component={React组件}**

这种情形下 ，component属性值可以是任何自定义的组件或者从其他第三方库导入的React组件。定义组件的代码请参考：

1. *// MyCustomInput.js*
2. import React, { Component } from 'react'
4. class MyCustomInput extends Component {
5. render() {
6. const { input: { value, onChange } } = this.props
7. return (
8. <div>
9. <span>The current value is {value}.</span>
10. <button type="button" onClick={() => onChange(value + 1)}>Inc</button>
11. <button type="button" onClick={() => onChange(value - 1)}>Dec</button>
12. </div>
13. )
14. }
15. }

然后在你的表单组件定义代码中便可以这样使用：

1. import MyCustomInput from './MyCustomInput'
3. ...
5. <Field name="myField" component={MyCustomInput}/>

**使用方式二：component={无状态函数}**

这是使用 <Field/> 的最灵活的方法，因为使用这种方法可以使你完全控制表单输入组件的渲染方式。而且，这种方式**对于显示校验错误信息特别有用**。当然，这种使用思路对于从以前版本的 redux-form使用转移过来的程序员来说是十分熟悉的。

【切记】**必须在render() 方法外部定义上述无状态函数**；否则，它会随着每次渲染都会被重新创建，从而由于组件的 prop发生了变化而使得系统强制 <Field/> 重新渲染。如果你在 render() 方法内部定义无状态函数，这不但会拖慢你的app，而且input组件每次都会在组件重新渲染的时候失去焦点。

1. // outside your render() method
2. const renderField = (field) => (
3. <div className="input-row">
4. <input {...field.input} type="text"/>
5. {field.meta.touched && field.meta.error &&
6. <span className="error">{field.meta.error}</span>}
7. </div>
8. )
10. // inside your render() method
11. <Field name="myField" component={renderField}/>

**使用方式三：component=“input”或者component=“select”或者component=“textarea”**

这种使用情况比较简单，比如创建一个文字输入框组件，你只需要使用如下方式：

&lt;Field component="input" type="text"/&gt;

易见，这种方式是把传统DOM元素用<Field/>API稍微一封装，并指明相应的type属性类型即可。

**一个复杂的例子**

说明：本例中的代码大部分来自于官方网站提供的redux-form-field-level-validation.js这个文件，也就是有关于在一个redux-form表单中进行按字段校验的情况。但是，有几句代码作了修改，有兴趣的同学请认真观察分析（为了便于参考，我把这个文件的修改版本的完整代码列举如下）：

1. import React from 'react'
2. import { Field, reduxForm } from 'redux-form'
4. const required = value => value ? undefined : 'Required'
5. const maxLength = max => value =>
6. value && value.length > max ? `Must be ${max} characters or less` : undefined
7. const maxLength15 = maxLength(15)
8. const number = value => value && isNaN(Number(value)) ? 'Must be a number' : undefined
9. const minValue = min => value =>
10. value && value < min ? `Must be at least ${min}` : undefined
11. const minValue18 = minValue(18)
12. const email = value =>
13. value && !/^[A-Z0-9.\_%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$/i.test(value) ?
14. 'Invalid email address' : undefined
15. const tooOld = value =>
16. value && value > 65 ? 'You might be too old for this' : undefined
17. const aol = value =>
18. value && /.+@aol\.com/.test(value) ?
19. 'Really? You still use AOL for your email?' : undefined
21. const renderField = ({ input, label, type, custom001,meta: { touched, error, warning } }) => (
22. <div>
23. <div>
24. <span>input</span>
25. <pre>{JSON.stringify(custom001)}</pre>
26. </div>
27. <label>{label}</label>
28. <div>
29. <input {...input} placeholder={label} type={type}/>
30. {touched && ((error && <span>{error}</span>) || (warning && <span>{warning}</span>))}
31. </div>
32. </div>
33. )
34. const selects = [
35. {
36. text:'-- Select --',
37. value:'-1'
38. },
39. {
40. text:'Red',
41. value:'ff0000'
42. },
43. {
44. text:'Green',
45. value:'00ff00'
46. },
47. {
48. text:'Blue',
49. value:'0000ff'
50. }
51. ]
53. const selectField = ({
54. input,
55. label,
56. selects,
57. meta: { touched, error, warning }
58. }) => (
59. <div>
60. <div>
61. <span>{label}</span>
62. <select {...input}>
63. {
64. selects.map((item, i) => (
65. <option key={i} value={item.value}>{item.text}</option>
66. ))
67. }
68. </select>
69. </div>
70. {touched &&
71. ((error && <div>{error}</div>) ||
72. (warning && <div>{warning}</div>))}
73. </div>
74. )
75. const FieldLevelValidationForm = (props) => {
76. const { handleSubmit, pristine, reset, submitting } = props
77. return (
78. <form onSubmit={handleSubmit}>
79. <Field name="username" type="text" custom001="custom00001" custom002="custom00002"
80. component={renderField} label="Username"
81. validate={[ required, maxLength15 ]}
82. />
83. <Field name="email" type="email" custom03="custom00003" custom04="custom00004"
84. component={renderField} label="Email"
85. validate={email}
86. warn={aol}
87. />
88. <Field name="age" type="number"
89. component={renderField} label="Age"
90. validate={[ required, number, minValue18 ]}
91. warn={tooOld}
92. />
94. <Field validate={[required]} component={selectField} selects={selects} label="Favorite Color" name="favoriteColor"/>
96. <div>
97. <button type="submit" disabled={submitting}>Submit</button>
98. <button type="button" disabled={pristine || submitting} onClick={reset}>Clear Values</button>
99. </div>
100. </form>
101. )
102. }
104. export default reduxForm({
105. form: 'fieldLevelValidation' // a unique identifier for this form
106. })(FieldLevelValidationForm)

上述代码中请注意如下几点：  
（1）<Field/>组件中仍然需要注意component属性的用法，还有自定义属性selects的用法；  
（2）因为上面代码的核心是讨论逐字段校验技术的，所以，仍然需要关注<Field/>组件中validate属性的表达方式；  
（3）再需要关注的是<Field/>组件中custom001等几个属性的用法（这里的表达毫无意义，只是概念探讨）。然后，在上面的renderField这个无状态函数（component属性的用法之一，对不对？）相关代码中，其参数使用了如下方式：  
{ input, label, type, custom001,meta: { touched, error, warning } }  
有关input和meta，官方网站已经提供了细致的说明，label和type对应于原始DOM标记的有关属性，而custom001则是上面代码中定义<Field/>组件时加入的一个定制属性(prop），这里通过下面的简单的测试方式提供了这个定制prop在<Field/>组件component属性值（是一个无状态函数）中的使用方法。

**小结**

在redux-form中，<Field/>的位置举足轻重，在本文相关的字段级校验用法中，难点在于对其component不同属性值（根据上面说明，可能是组件，无状态函数或者常规DOM字符串）的属性的理解，还有对于validate这个属性值表达方式的理解。总体感觉，一下彻底掌握redux-form也不是一件容易事。总之，如果选择了React，那么你会在这个“泥沼”中“滞留”很长一段时间。