**[React组件之间传值](http://blog.csdn.net/limm33/article/details/50942863)**

标签： [React](http://www.csdn.net/tag/React)[组件](http://www.csdn.net/tag/%e7%bb%84%e4%bb%b6)[传值](http://www.csdn.net/tag/%e4%bc%a0%e5%80%bc)[交互](http://www.csdn.net/tag/%e4%ba%a4%e4%ba%92)

2016-03-21 10:07 9554人阅读 [评论](http://blog.csdn.net/limm33/article/details/50942863#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/limm33/article/details/50942863#report)

本文章已收录于：

[[](http://lib.csdn.net/base/24?source=blogtop)***React知识库***](http://lib.csdn.net/base/24?source=blogtop)

http://static.blog.csdn.net/images/category_icon.jpg分类：

>>React*（39）* http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

目录[(?)](http://blog.csdn.net/limm33/article/details/50942863" \o "系统根据文章中H1到H6标签自动生成文章目录)[[+]](http://blog.csdn.net/limm33/article/details/50942863)

前言

今天群里面有很多都在问关于 React 组件之间是如何通信的问题，之前自己写的时候也遇到过这类问题。下面是我看到的一篇不错英文版的翻译，看过我博客的人都知道，我翻译可能不会按部就班，会尽可能用中文的意思，来将作者要讲述的技术描述清楚。英文能力有限，如果有不对的地方请跟我留言，一定修改……^\_^

原著序

处理 [**React**](http://lib.csdn.net/base/react) 组件之间的交流方式，主要取决于组件之间的关系，然而这些关系的约定人就是你。

我不会讲太多关于 data-stores、data-adapters 或者 data-helpers 之类的话题。我下面只专注于 React 组件本身的交流方式的讲解。

React 组件之间交流的方式，可以分为以下 3 种：

* 【父组件】向【子组件】传值；
* 【子组件】向【父组件】传值；
* 没有任何嵌套关系的组件之间传值（PS：比如：兄弟组件之间传值）

一、【父组件】向【子组件】传值

初步使用

这个是相当容易的，在使用 React 开发的过程中经常会使用到，主要是利用 props 来进行交流。例子如下：

// 父组件

var MyContainer = React.createClass({

getInitialState: function () {

return {

checked: true

};

},

render: function() {

return (

<ToggleButton text="Toggle me" checked={this.state.checked} />

);

}

});

// 子组件

var ToggleButton = React.createClass({

render: function () {

// 从【父组件】获取的值

var checked = this.props.checked,

text = this.props.text;

return (

<label>{text}: <input type="checkbox" checked={checked} /></label>

);

}

});

进一步讨论

如果组件嵌套层次太深，那么从外到内组件的交流成本就变得很高，通过 props 传递值的优势就不那么明显了。（PS：所以我建议尽可能的减少组件的层次，就像写 HTML 一样，简单清晰的结构更惹人爱）

// 父组件

var MyContainer = React.createClass({

render: function() {

return (

<Intermediate text="where is my son?" />

);

}

});

// 子组件1：中间嵌套的组件

var Intermediate = React.createClass({

render: function () {

return (

<Child text={this.props.text} />

);

}

});

// 子组件2：子组件1的子组件

var Child = React.createClass({

render: function () {

return (

<span>{this.props.text}</span>

);

}

});

二、【子组件】向【父组件】传值

接下来，我们介绍【子组件】控制自己的 state 然后告诉【父组件】的点击状态，然后在【父组件】中展示出来。因此，我们添加一个 change 事件来做交互。

// 父组件

var MyContainer = React.createClass({

getInitialState: function () {

return {

checked: false

};

},

onChildChanged: function (newState) {

this.setState({

checked: newState

});

},

render: function() {

var isChecked = this.state.checked ? 'yes' : 'no';

return (

<div>

<div>Are you checked: {isChecked}</div>

<ToggleButton text="Toggle me"

initialChecked={this.state.checked}

callbackParent={this.onChildChanged}

/>

</div>

);

}

});

// 子组件

var ToggleButton = React.createClass({

getInitialState: function () {

return {

checked: this.props.initialChecked

};

},

onTextChange: function () {

var newState = !this.state.checked;

this.setState({

checked: newState

});

// 这里要注意：setState 是一个异步方法，所以需要操作缓存的当前值

this.props.callbackParent(newState);

},

render: function () {

// 从【父组件】获取的值

var text = this.props.text;

// 组件自身的状态数据

var checked = this.state.checked;

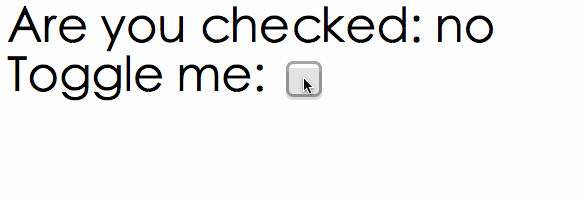
return (

<label>{text}: <input type="checkbox" checked={checked} onChange={this.onTextChange} /></label>

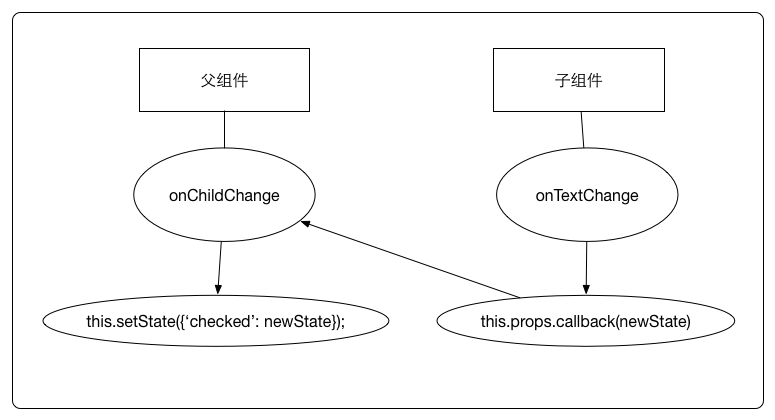
);

}

});



我觉得原文作者用代码不是很直观，接下来我话一个流程走向简图来直观描述一下这个过程：



这样做其实是依赖 props 来传递事件的引用，并通过回调的方式来实现的，这样实现不是特别好，但是在没有任何工具的情况下也是一种简单的实现方式

这里会出现一个我们在之前讨论的问题，就是组件有多层嵌套的情况下，你必须要一次传入回调函数给 props 来实现子组件向父组件传值或者操作。

Tiny-Tip: React Event System

在 onChange 事件或者其他 React 事件中，你能够获取以下东西：

* 【this】：指向你的组件
* 【一个参数】：这个参数是一个 [React 合成事件](http://wiki.jikexueyuan.com/project/react/event-system.html)，SyntheticEvent。

React 对所有事件的管理都是自己实现的，与我们之前使用的 onclick、onchange 事件不一样。从根本上来说，他们都是绑定到 body 上。

document.on('change', 'input[data-reactid=".0.2"]', function () {...});

上面这份代码不是来自于 React，只是打一个比方而已。

如果我没有猜错的话，React 真正处理一个事件的代码如下：

var listenTo = ReactBrowserEventEmitter.listenTo;

...

function putListener(id, registrationName, listener, transaction) {

...

var container = ReactMount.findReactContainerForID(id);

if (container) {

var doc = container.nodeType === ELEMENT\_NODE\_TYPE ? container.ownerDocument : container;

listenTo(registrationName, doc);

}

...

}

// 在监听事件的内部，我们能发现如下：

target.addEventListener(eventType, callback, false);

这里有所有 React 支持的事件：[中文文档-事件系统](http://reactjs.cn/react/docs/events.html" \t "_blank)

多个子组件使用同一个回调的情况

// 父组件

var MyContainer = React.createClass({

getInitialState: function () {

return {

totalChecked: 0

};

},

onChildChanged: function (newState) {

var newToral = this.state.totalChecked + (newState ? 1 : -1);

this.setState({

totalChecked: newToral

});

},

render: function() {

var totalChecked = this.state.totalChecked;

return (

<div>

<div>How many are checked: {totalChecked}</div>

<ToggleButton text="Toggle me"

initialChecked={this.state.checked}

callbackParent={this.onChildChanged}

/>

<ToggleButton text="Toggle me too"

initialChecked={this.state.checked}

callbackParent={this.onChildChanged}

/>

<ToggleButton text="And me"

initialChecked={this.state.checked}

callbackParent={this.onChildChanged}

/>

</div>

);

}

});

// 子组件

var ToggleButton = React.createClass({

getInitialState: function () {

return {

checked: this.props.initialChecked

};

},

onTextChange: function () {

var newState = !this.state.checked;

this.setState({

checked: newState

});

// 这里要注意：setState 是一个异步方法，所以需要操作缓存的当前值

this.props.callbackParent(newState);

},

render: function () {

// 从【父组件】获取的值

var text = this.props.text;

// 组件自身的状态数据

var checked = this.state.checked;

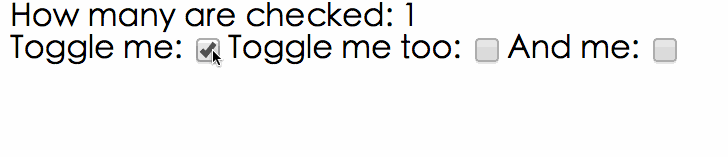
return (

<label>{text}: <input type="checkbox" checked={checked} onChange={this.onTextChange} /></label>

);

}

});



这是非常容易理解的，在父组件中我们增加了一个【totalChecked】来替代之前例子中的【checked】，当子组件改变的时候，使用同一个子组件的回调函数给父组件返回值。

三、没有任何嵌套关系的组件之间传值

如果组件之间没有任何关系，组件嵌套层次比较深（个人认为 2 层以上已经算深了），或者你为了一些组件能够订阅、写入一些信号，不想让组件之间插入一个组件，让两个组件处于独立的关系。对于事件系统，这里有 2 个基本操作步骤：订阅（subscribe）/监听（listen）一个事件通知，并发送（send）/触发（trigger）/发布（publish）/发送（dispatch）一个事件通知那些想要的组件。

下面讲介绍 3 种模式来处理事件，你能[点击这里](https://github.com/millermedeiros/js-signals/wiki/Comparison-between-different-Observer-Pattern-implementations" \t "_blank)来比较一下它们。

简单总结一下：

(1) Event Emitter/Target/Dispatcher

特点：需要一个指定的订阅源

// to subscribe

otherObject.addEventListener(‘click’, function() { alert(‘click!’); });

// to dispatch

this.dispatchEvent(‘click’);

(2) Publish / Subscribe

特点：触发事件的时候，你不需要指定一个特定的源，因为它是使用一个全局对象来处理事件（其实就是一个全局  
广播的方式来处理事件）

// to subscribe

globalBroadcaster.subscribe(‘click’, function() { alert(‘click!’); });

// to dispatch

globalBroadcaster.publish(‘click’);

(3) Signals

特点：与Event Emitter/Target/Dispatcher相似，但是你不要使用随机的字符串作为事件触发的引用。触发事件的每一个对象都需要一个确切的名字（就是类似硬编码类的去写事件名字），并且在触发的时候，也必须要指定确切的事件。（看例子吧，很好理解）

// to subscribe

otherObject.clicked.add(function() { alert(‘click’); });

// to dispatch

this.clicked.dispatch();

如果你只想简单的使用一下，并不需要其他操作，可以用简单的方式来实现：

// 简单实现了一下 subscribe 和 dispatch

var EventEmitter = {

\_events: {},

dispatch: function (event, data) {

if (!this.\_events[event]) { // 没有监听事件

return;

}

for (var i = 0; i < this.\_events[event].length; i++) {

this.\_events[event][i](data);

}

},

subscribe: function (event, callback) {

// 创建一个新事件数组

if (!this.\_events[event]) {

this.\_events[event] = [];

}

this.\_events[event].push(callback);

}

};

otherObject.subscribe('namechanged', function(data) { alert(data.name); });

this.dispatch('namechanged', { name: 'John' });

如果你想使用 Publish/Subscribe 模型，可以使用：[PubSubJS](https://github.com/mroderick/PubSubJS" \t "_blank)

React 团队使用的是：[js-signals](http://millermedeiros.github.io/js-signals/" \t "_blank) 它基于 Signals 模式，用起来相当不错。

Events in React

使用 React 事件的时候，必须关注下面两个方法：

componentDidMount

componentWillUnmount

在处理事件的时候，需要注意：

在 componentDidMount 事件中，如果组件挂载（mounted）完成，再订阅事件；当组件卸载（unmounted）的时候，在 componentWillUnmount 事件中取消事件的订阅。

（如果不是很清楚可以查阅 React 对生命周期介绍的文档，里面也有描述。原文中介绍的是 componentWillMount 个人认为应该是挂载完成后订阅事件，比如Animation这个就必须挂载，并且不能动态的添加，谨慎点更好）

因为组件的渲染和销毁是由 React 来控制的，我们不知道怎么引用他们，所以EventEmitter 模式在处理组件的时候用处不大。

pub/sub 模式可以使用，你不需要知道引用。

下面来一个例子：实现有多个 product 组件，点击他们的时候，展示 product 的名字。

(我在例子中引入了之前推荐的 PubSubJS 库，如果你觉得引入代价太大，也可以手写一个简版，还是比较容易的，很好用哈，大家也可以体验，但是我还是不推荐全局广播的方式)

// 定义一个容器

var ProductList = React.createClass({

render: function () {

return (

<div>

<ProductSelection />

<Product name="product 1" />

<Product name="product 2" />

<Product name="product 3" />

</div>

);

}

});

// 用于展示点击的产品信息容器

var ProductSelection = React.createClass({

getInitialState: function() {

return {

selection: 'none'

};

},

componentDidMount: function () {

this.pubsub\_token = PubSub.subscribe('products', function (topic, product) {

this.setState({

selection: product

});

}.bind(this));

},

componentWillUnmount: function () {

PubSub.unsubscribe(this.pubsub\_token);

},

render: function () {

return (

<p>You have selected the product : {this.state.selection}</p>

);

}

});

var Product = React.createClass({

onclick: function () {

PubSub.publish('products', this.props.name);

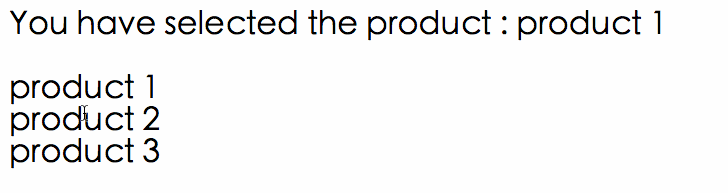
},

render: function() {

return <div onClick={this.onclick}>{this.props.name}</div>;

}

});



ES6: yield and js-csp

ES6 中有一种传递信息的方式，使用生成函数(generators)和 yield 关键字。可以看一下[https://github.com/ubolonton/js-csp](https://github.com/ubolonton/js-csp" \t "_blank)

（这里我写一个简单的 DEMO 介绍一下这种新的传递方式，其实大同小异）

function\* list() {

for(var i = 0; i < arguments.length; i++) {

yield arguments[i];

}

return "done.";

}

var o = list(1, 2, 3);

var cur = o.next;

while(!cur.done) {

cur = o.next();

console.log(cur);

}

以上例子来自于屈屈的一篇博客：[ES6 中的生成器函数介绍](http://imququ.com/post/generator-function-in-es6.html" \t "_blank) 屈屈是一个大牛，大家可以经常关注他的博客。

通常来说，你有一个队列，对象在里面都能找到一个引用，在定义的时候锁住，当发生的时候，立即打开锁执行。js-csp 是一种解决办法，也许以后还会有其他解决办法。

结尾

在实际应用中，按照实际要解决的需求选择解决办法。对于小应用程序，你可以使用 props 和回调的方法进行组件之间的数据交换。你可以通过 pub/sub 模式，以避免污染你的组件。在这里，我们不是在谈论数据，只是组件。对于数据的请求、数据的变化等场景，可以使用 Facebook 的 Flux、Relay、GraphQL 来处理，都非常的好用。

文中的每一个例子我都验证过了，主要使用最原始的引入文件方式，创建服务使用的 http-server 包，大家也可以尝试自己来一次。