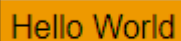


# React 开发实战

## 1 Tips

### 1.1 内联样式 自动追加正确的单位

```
1 import React, { Component } from 'react'
2
3 class Stylespan extends Component {
4   // 可以自动追加正确的单位
5   render () {
6     let divStyle = {
7       width: 100,
8       height: 30,
9       padding: 5,
10      backgroundColor: '#ee9900'
11    }
12
13    return (
14      <div style={ divStyle }>
15        Hello world
16      </div>
17    )
18  }
19 }
20
21
22 export default Stylespan
```



### 1.2 受控组件和非受控组件

- 受控组件

```
1 // 受控组件 值由react 控制
2 // 通过handleChangeInput 来更新组件
3 <input type="search" value={ this.state.searchTerm } onChange={ e => {
4   this.handleChangeInput(e)
5 } } />
```

- 非受控组

不为react 控制的组件

```
1 import React, { Component } from 'react'
2
3 class Uncontrolled extends Component {
4   constructor () {
5     super()
6     this.state = {
```

```

7     name: '赵思',
8     email: 'magicwingzs@qq.com'
9   }
10 }
11 handleSubmit (e) {
12   e.preventDefault()
13   console.dir(e.target.name.value)
14   console.dir(e.target.Email.value)
15 }
16 handleChangeInput (e) {
17   console.log(e.target.value)
18 }
19 render () {
20   return (
21     <form onSubmit={ this.handleSubmit.bind(this) }>
22       <div>
23         /* 通过defaultValue 来设置默认值 */
24         Name: <input name="name" type="text" defaultValue={
this.state.name } onChange={ e => this.handleChangeInput(e) } />
25       </div>
26       <div>
27         Email: <input name="Email" type="email" defaultValue={
this.state.email } />
28       </div>
29       <button type="submit">
30         Submit
31       </button>
32     </form>
33   )
34 }
35 }
36
37
38 export default Uncontrolled

```

## 1.3 自定义propTypes 校验器

```

1  import React, { Component } from 'react'
2  import PropTypes from 'prop-types'
3
4  // 自定义的校验器
5  let titlePropType = (props, propName, componentName) => {
6    if (props[propName]) {
7      let value = props[propName]
8      if (typeof value !== 'string' || value.length > 80) {
9        return new Error(
10          `${propName} in ${componentName} is longer then 80 characters`
11        )
12      }
13    }
14  }
15
16
17  class Card extends Component {
18    static propTypes = {

```

```

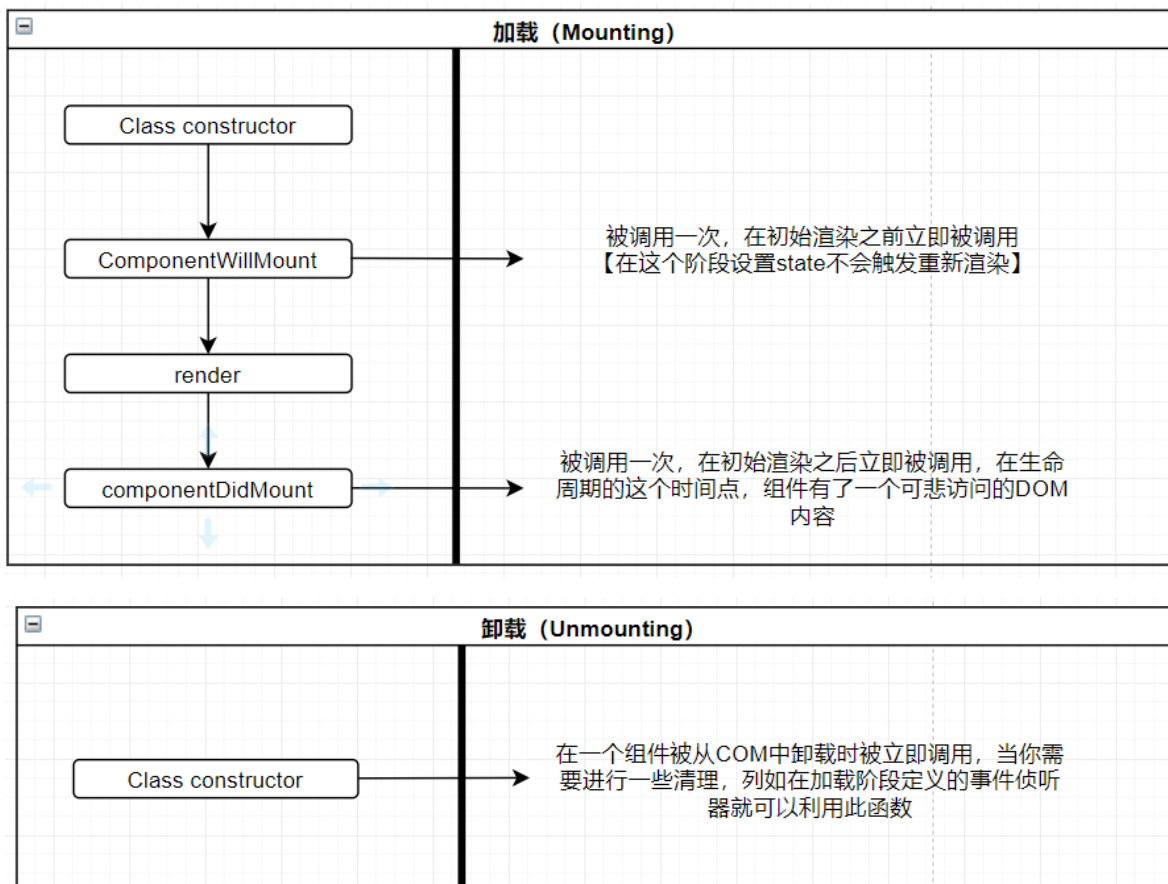
19     id: PropTypes.number,
20     // 使用自定义的校验
21     title: titlePropType,
22     description: PropTypes.string,
23     color: PropTypes.string,
24     tasks: PropTypes.arrayOf(PropTypes.object)
25   }
26   constructor () {
27     super()
28     this.state = {
29     }
30   }
31   render () {
32     return (
33       <div className="card">
34       </div>
35     )
36   }
37 }
38
39 export default Card

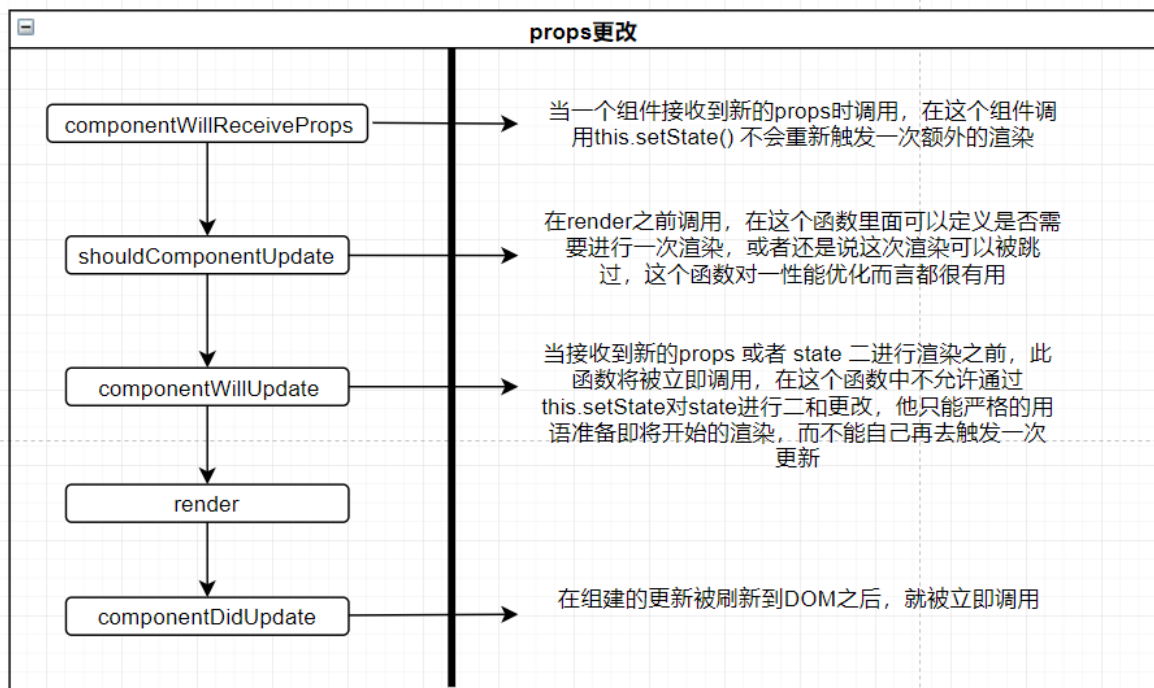
```

验证失败时

Warning: Failed prop type: title in Card is longer then 80 characters [index.js:1375](#)  
 in Card (at List.js:13)  
 in List (at KanbanBoard.js:15)  
 in div (at KanbanBoard.js:14)  
 in KanbanBoard (at src/index.js:14)

## 1.4 组件的生命周期





## 1.5 快捷调用传递下来的函数

```

1  let tasks = this.props.tasks.map((task, index) => {
2      return (
3          <li className="checklist__task" key={ task.id }>
4              <input type="checkbox" checked={ task.done }
5                  // 调用传递下来的函数
6                  onChange={ this.props.taskCallbacks.toggle.bind(null,
7                      this.props.cardId, task.id, index) }
8              { task.name }
9              { /* 删除任务 */ }
10             <i className="checklist__task--remove" />
11         </li>
12     )
13 }
  
```

## 1.6 键盘事件 回车事件

## 2 浅谈不可变性

基本概念：我们不去直接修改一个对象，我们直接替换它

### 2.1 性能取决

比较和检查一个JavaScript 对象是否被修改过

如果在对象被修改时，不是直接修改它，而是直接将它替换点，那么查看对象是否被修改就会很快

```
1  import React, { Component } from 'react'
2
3  class Voucher extends Component {
4    constructor () {
5      super()
6      this.state = {
7        passengers: [
8          'Simmon, Robert A',
9          'Taylor, Kathleen R'
10       ],
11       ticket: {
12         company: 'Dalta',
13         filghtNo: '0990',
14         departure: {
15           airport: 'LAS',
16           time: '2016-08-27'
17         },
18         arrival: {
19           airport: 'MIA1',
20           time: '2016-08-21'
21         },
22         codeshare: [
23           {
24             company: 'GL',
25             filghtNo: '9840'
26           },
27           {
28             company: 'TM',
29             filghtNo: '5010'
30           }
31         ]
32       }
33     }
34   }
35   render () {
36     return (
37       <div>
38
39       </div>
40     )
41   }
42 }
43
44
45 export default Voucher
```

## 2.2 concat 修改数组

假设现在想要将一个新城可添加到passengers 数组中，不小心会直接修改组件的state对象

```
1 let updatedPassengers = this.state.passengers;
2 updatedPassengers.push('Mitchell, Vincent M.')
3 this.setState({
4   passengers: updatedPassengers
5 })
```

创建updatedPassengers并不是创建了一个副本，而是创建了一个新引用

下面使用Array的concat方法 非入侵式添加了一个新乘客

```
1 let updatedPassengers = this.state.passengers.concat('Mitchell, Vincent M.')
2 this.setState({
3   passengers: updatedPassengers
4 })
```

## 2.3 Object.assign 修改对象

```
1 Object.assign(target, source_1, .... , source_n)
2 // 首先遍历source_1 对象的所有属性 将他们赋值到target 对象上，后面同样操作
```

列入要修改ticket 的 flightNo

```
1 let updatedTick = Object.assign({}, this.state.ticket, { flightNo: '1010' })
2 this.setState({
3   ticket: updatedTick
4 })
```

## 2.4 嵌套对象

### 2.4.1 使用不变性助手

```
1 npm i react-addons-update -S
2
3 import update from 'react-addons-update'
```

表 3-4 React 不变性助手的命令

命令	描述
\$push	类似于 Array 的 push 函数，它向一个数组的尾部添加一个或多个元素。例如： <pre>let initialArray = [1, 2, 3]; let newArray = update(initialArray, {\$push: [4]}); // =&gt; [1, 2, 3, 4]</pre>
\$unshift	类似于 Array 的 unshift 函数，它在一个数组的头部添加一个或多个元素。例如： <pre>let initialArray = [1, 2, 3]; let newArray = update(initialArray, {\$unshift: [0]}); // =&gt; [0, 1, 2, 3]</pre>
\$splice	类似于 Array 的 splice 函数，它通过从数组中移除元素且/或向数组中添加元素，来修改一个数组的内容。和 Array.splice 主要的语法区别是你需要提供一个元素为数组的数组来作为参数，作为参数的数组中的每一个数组包含了对数组进行 splice 操作的参数。例如： <pre>let initial Array = [1, 2, 'a']; let newArray = update(initialArray, {\$splice: [[2,1,3,4]]}); // =&gt; [1, 2, 3, 4]</pre>
\$set	完整地替换掉整个目标
\$merge	将给定对象的键合并到目标对象中。例如： <pre>let ob. = {a: 5, b: 3}; let newObj = update(obj, {\$merge: {b: 6, c: 7}}); // =&gt; {a: 5, b: 6, c: 7}</pre>
\$apply	将当前的值传给一个函数，在函数中对传入的值进行修改，然后使用函数的返回值作为结果。例如： <pre>let obj = {a: 5, b: 3}; let newObj = update(obj, {b: {\$apply: (value) =&gt; value*2 }}); // =&gt; {a: 5, b: 6}</pre>

update 接受两个参数 第一个参数是你想要更新的对象或者数组，第二个参数是一个对象，他描述了你想要在何处进行何种修改

```

1  let student = { name: 'John Caster', grades: { 'A', 'C', 'B' } }
2
3  // 数组新增一个元素
4  // 定位到grader属性 进行push 修改
5  注意： 一个对象中的数组需要向下方这样写
6  let newSyudent = update(student, { grades: { $push: ['A'] } })
7
8  正确的写法
9  this.setState(update(this.state, {
10    goods: {
11      $push: [{
12        id: this.state.goods.length + 1,
13        name: this.state.text
14      }]
15    }
16  }))
17
18  错误的写法
19  this.setState(update(this.state.goods, {
20    $push: [{
21      id: this.state.goods.length + 1,
```

```

22     name: this.state.text
23   }]
24 })
25
26 // 如果想要完全修改整个数组 可以用$set 命令
27 let newStudent = update(student, { grades: { $set: ['a', 'a', 'b'] } })

```

想要修改上面机票对象总的arrival中的airport

```

1  let originTicket = {
2    company: 'Dalta',
3    flightNo: '0990',
4    departure: {
5      airport: 'LAS',
6      time: '2016-08-27'
7    },
8    arrival: {
9      airport: 'MIA1',
10     time: '2016-08-21'
11   },
12   codeshare: [
13     {
14       company: 'GL',
15       flightNo: '9840'
16     },
17     {
18       company: 'TM',
19       flightNo: '5010'
20     }
21   ]
22 }
23
24 let newTicket = update(originTicket, {
25   arrival: {
26     airport: { $set: 'MCO' }
27   }
28 })

```

#### 2.4.1.1 例子

```

1  [
2    {
3      "id": 11977,
4      "title": "Read the Book",
5      "description": "I should read the **whole** book",
6      "color": "#BD8D31",
7      "status": "in-progress",
8      "tasks": []
9    },
10   {
11     "id": 11978,
12     "title": "Write some code",
13     "description": "Code along with the samples in the book at [github]
14     (https://github.com/pro-react),
15     "color": "#3A7E28",
16     "status": "todo",

```



```

16     "tasks": [
17         {
18             "id": 42399,
19             "name": "ContactList Example",
20             "done": true
21         },
22         {
23             "id": 42400,
24             "name": "Kanban Example",
25             "done": false
26         },
27         {
28             "id": 42401,
29             "name": "My own experiments",
30             "done": false
31         }
32     ]
33 }
34 ]

```

现需要删除某一个card 中的任务列表中某一任务

```

1  // 删除任务
2  deleteTask (cardId, taskId, taskIndex) {
3      // 通过卡片id 拿到 卡片索引
4      let cardIndex = this.state.cards.findIndex(card => card.id === cardId)
5      // 创建新的副本
6      let newCards = update(this.state.cards, {
7          // 通过index 定位
8          [cardIndex]: {
9              // 进行删除操作
10             tasks: { $splice: [ [taskIndex, 1] ] }
11         }
12     })
13 }

```

操作数组

```

1  updateCardPosition (cardId, afterId) {
2      if (cardId !== afterId) {
3          // 获取当前卡片Index
4          let cardIndex = this.state.cards.findIndex((card) => card.id ===
cardId)
5          // 获取当前卡片对象
6          let card = this.state.cards[cardIndex]
7
8          // 获取下一个列表索引
9          let afterIndex = this.state.cards.findIndex((cardId) => card.id ==
afterId)
10
11         this.setState(update(this.state, {
12             cards: {
13                 $splice: [
14                     [cardIndex, 1],
15                     [afterIndex, 0 ,card]
16                 ]

```

```

17     }
18     )))
19 }

```

## 修改对象值

```

1  updateCardStatus (cardId, listId) {
2    // 获取卡片在当前列表的索引
3    let cardIndex = this.state.cards.findIndex((card) => card.id ===
cardId)
4    // 获取当前的卡片
5    let card = this.state.cards[cardIndex]
6    // 如果当前卡片不等于传入的卡片列表的话
7    if (card.status !== listId) {
8      // 使用不可变性助手、
9      this.setState(update(this.state, {
10        cards: {
11          [cardIndex]: {
12            status: {$set: listId}
13          }
14        }
15      })))
16    }
17  }

```

```

1  // 购物车列表
2  cartList: []
3
4  // 加入到购物车
5  addCard (good) {
6    let goodIndex = this.state.cartList.findIndex(item => {
7      return item.id === good.id
8    });
9    if (goodIndex !== -1) {
10     this.setState(update(this.state, {
11       cartList: {
12         [goodIndex]: {
13           count: {
14             $apply: (value) => value + 1
15           }
16         }
17       }
18     })))
19   } else {
20     this.setState(update(this.state, {
21       cartList: {
22         $push: [{
23           ...good,
24           count: 1
25         }]
26       }
27     })))
28   }
29 };

```

