react 它其实就是一个和数学函数差不多的，xyz 在已经知道 x y 的情况下求出z

特点：

1. 单项的数据流形式
2. 一个用于创建 可复用 可聚合的web框架
3. 将js逻辑于html标签紧紧结合，通俗易懂

4.一点状态state改变，重新渲染整个dom

渲染方式：用户输入--》从API获取数据-->将数据传给顶层组件---》react将每个组件渲染出来

不在拥有 魔法般的双向数据绑定

1. 每个组件都是干啥的，很直观 结果更加可以预测 组件之间的关系更清晰
2. 独立的函数 接受参数 ，可复用，可传递，返回结果渲染组件
3. 虚拟的Dom tree
4. React 重建dom树
5. 找到上个版本的DOM的差异
6. 计算出最新的DOM更新错做
7. 从操作的队列进行批量的进行dom操作
8. 服务端和客户端共用逻辑（少写代码）
9. 直接渲染特定的页面而不用渲染整个app

https://facebook.github.io/react/downloads.html 下

$npm show bower 查看当前的bower环境

$npm install -g bower

state render ruselt 单数据流，和数学函数一样 x y z 在已知 x y 求z

组件声明时：

1.getDefaultProps - > 获取默认值

2.getInitialState -> 获取初始的状态

3.componentWillMount ->组件即将更新装载

4.render -> 进行装载，做媒

5.componentDidMount ->组件装载完毕

组件的更新过程：

1.componentRecieveProps ->父组件-子组件的数据(props中间的数据)传递

2.shouldComponentUpdate->检查组件是否需要更新

3.componentWillUpdate->组件即将更新

4.render

5.componentDidUpdate

6.componentWillUnmount->即将卸载

7.componentDidUnmount->卸载完毕

getInitialState

object getInitialState()

该组件被安装之前调用一次。返回值将被用作初始值this.state。

注意：

这种方法不适用于ES6 class扩展组件React.Component。欲了解更多信息，请参阅有关文档ES6类。

getDefaultProps

object getDefaultProps()

调用一次，并在创建类时缓存。在映射值将被设置this.props（即使用一个如果不是由父组件指定的道具被in检查）。

在创建任何实例之前，此方法被调用，因此不能依靠this.props。

此外，要注意，在返回的任何复杂的对象getDefaultProps()将在实例共享，不可复制。

MyComponent.propTypes = {

// You can declare that a prop is a specific JS primitive. By default, these

// are all optional.

optionalArray: React.PropTypes.array,

optionalBool: React.PropTypes.bool,

optionalFunc: React.PropTypes.func,

optionalNumber: React.PropTypes.number,

optionalObject: React.PropTypes.object,

optionalString: React.PropTypes.string,

optionalSymbol: React.PropTypes.symbol,

// Anything that can be rendered: numbers, strings, elements or an array

// (or fragment) containing these types.

optionalNode: React.PropTypes.node,

生命周期方法

各种方法在特定点在组件的生命周期中执行。

安装：componentWillMount

void componentWillMount()

调用一次，无论是客户端和服务器上，立即出现初始渲染之前。如果调用setState此方法中，

render()将看到更新的状态，将尽管状态变化只执行一次。

安装：componentDidMount

void componentDidMount()

调用一次，只有客户（而不是服务器）上，立即出现初始渲染后。在生命周期这一点上，

你可以访问任何裁判给你的孩子（如访问底层DOM表示）。该componentDidMount()子组件的方法，在此之前父组件的调用。

如果你想与其他JavaScript框架集成，使用定时器设置setTimeout，或setInterval，

或发送AJAX请求，在这个方法中执行这些操作。

更新：componentWillReceiveProps

void componentWillReceiveProps(

object nextProps

)

当组件接收新道具时调用。是不是调用此方法初始渲染。

以此为契机，以应对道具转换之前render()通过使用更新状态调用this.setState()。

旧的道具可以通过以下方式访问this.props。调用this.setState()这个函数中不会引发额外的渲染。

componentWillReceiveProps: function(nextProps) {

this.setState({

likesIncreasing: nextProps.likeCount > this.props.likeCount

});

}

注意：

一个常见的错误是这个生命周期方法的过程中执行的假设，道具已经改变代码。

要理解为什么这是无效的，读A蕴含B不意味着b。默示一个

没有类似的方法componentWillReceiveState。传入丙转变可导致状态改变，

但相反的是不正确的。如果需要在响应状态改变执行操作，使用componentWillUpdate。

更新：shouldComponentUpdate

boolean shouldComponentUpdate(

object nextProps, object nextState

)

正在接收新道具或状态时，呈现之前调用。这种方法不要求对初始呈现或时forceUpdate被使用。

以此为契机，return false当你确定新的道具和状态的转换将不需要组件更新。

shouldComponentUpdate: function(nextProps, nextState) {

return nextProps.id !== this.props.id;

}

如果shouldComponentUpdate返回false，那么render()将完全跳过，直到下一个状态变化。

此外，componentWillUpdate与componentDidUpdate将不被调用。

默认情况下，shouldComponentUpdate始终返回true防止微妙的错误时state到位突变，

但如果你仔细始终将state作为不可改变的，只读的props并且state在render()那么可以覆盖shouldComponentUpdate与旧的道具和状态进行比较，以接替他们的实现。

如果性能是一个瓶颈，尤其是在数十或数百个组件，使用shouldComponentUpdate以加快您的应用程序。

更新：componentWillUpdate

void componentWillUpdate(

object nextProps, object nextState

)

立即正在接收新道具或状态时，呈现之前调用。是不是调用此方法初始渲染。

以此为契机更新之前执行的准备。

注意：

你不能用this.setState()这种方法。如果你需要在响应道具的变化更新状态，

使用componentWillReceiveProps来代替。

更新：componentDidUpdate

void componentDidUpdate(

object prevProps, object prevState

)

组件的更新之后，立即调用刷新到DOM。是不是调用此方法初始渲染。

使用此当组件已经更新了DOM操作的机会。

正在卸载：componentWillUnmount

void componentWillUnmount()

立即调用之前的成分是从DOM卸载。

在此方法中执行任何必要的清理，如无效定时器或清理中创建的任何DOM元素componentDidMount。

react中的表单

不可控组件

<input type='text' defaultValue='hello world'/>

这样的组件我们称为不可控组件 数据被直接写死了，这个数据是存储在input中，组件中的数据和state中的数据

有可能不再对应

getInitiaState(){}

<input type='text' defaultValue={this.state.value}/>

var inputValue = this.state.value;

写入和获取都保持一致

复用事件处理函数

<input onChange={this.handleChange.bind(this,’input1’)}

<input onChange={this.handleChange2}

<input onChange={this.handleChange3}

<input onChange={this.handleChange4}

.................数据大的时候要写很多

常用的2个方法

handleChange：function(name,e){........}

<input onChange={this.handleChange.bind(this,’input1’)}

e.nativeEvent 真实的事件 注意这个不是跨浏览器的，每个浏览器的有差异