React

需要安装的东西：

node.js，尽量新的版本，16点多

npm，用来下载nodejs包的，有点像pip

查看npm版本：npm -v

更新npm：Linux：sudo npm install npm -g

Windows：npm install npm -g

npm更新的时候可能会报错，-4048什么的，说是系统阻止这么干

解决方法：**nodejs卸载了重装就行。。。**搜索一下有个uninstall node的程序

清除缓存、管理员模式打开cmd什么的都没用

重启/重装解决一切问题

安装完新版node和npm后又遇到点问题：

npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security updates. Please upgrade asap.

不更新tar的话，create-react-app没法运行

安装个新版tar就好：

npm i tar

然后又遇到问题，在创建项目的时候，'create-react-app' 不是内部或外部命令，也不是可运行的程序或批处理文件。

npm config list找到prefix，那个就是放全局包的地方（应该就是安装的时候写了-g的那种？）

然后这个路径可以改，在一个.npsrc的文件里面

把这个路径加入环境变量path里，就好了

我现在的是

安装东西：

npm install 包名称

JS

```
class Tweet
{
    state = {};
    render()
    {
        blabla
```

```
  }
}
```

Angular, Vue: framework

React: library

install:

```
npm install -g create-react-app
```

create:

create-react-app

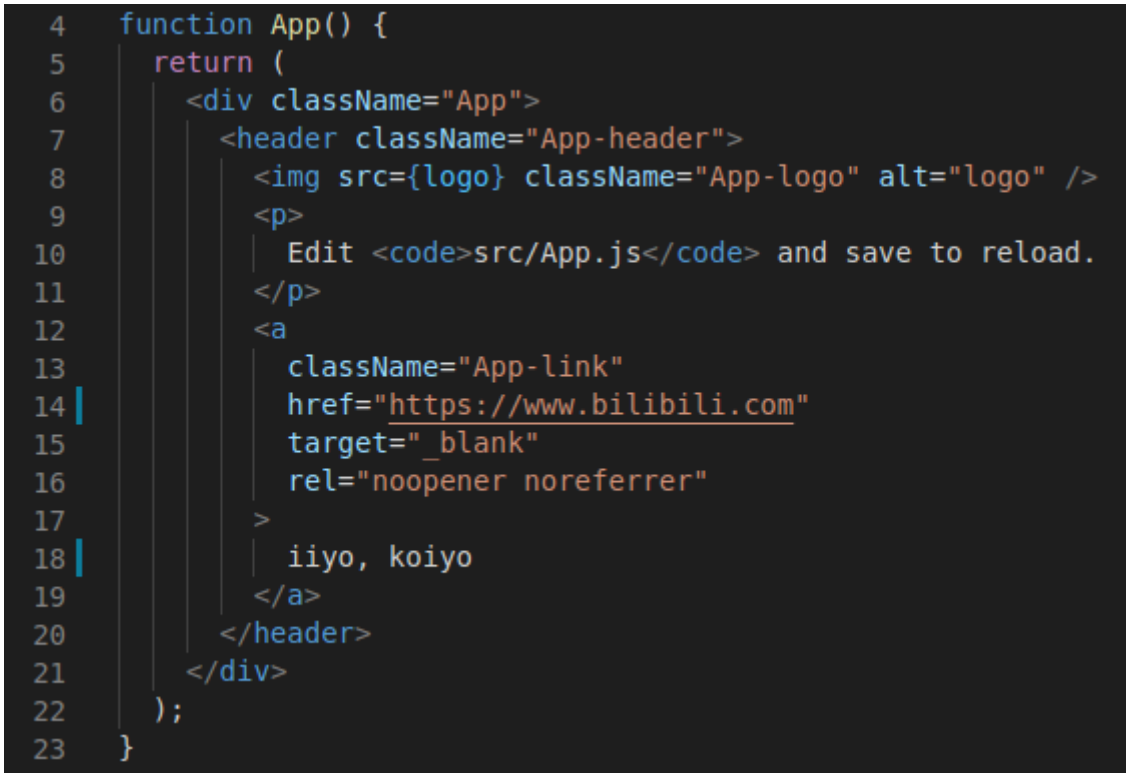
```
(cd work_directory)
create-react-app mingzi
cd mingzi
```

start :

```
npm start
```

render: return a markup

e.g.default code



```
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://www.bilibili.com"
          target="_blank"
          rel="noopener noreferrer"
        >
          iiyo, koiyo
        </a>
      </header>
    </div>
  );
}
```

babel: online js compiler

babeljs.io

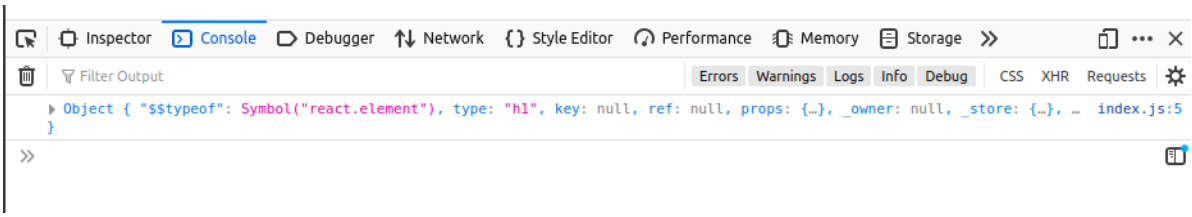
compile react: babeljs.io/repl

IIYO, KOIYO

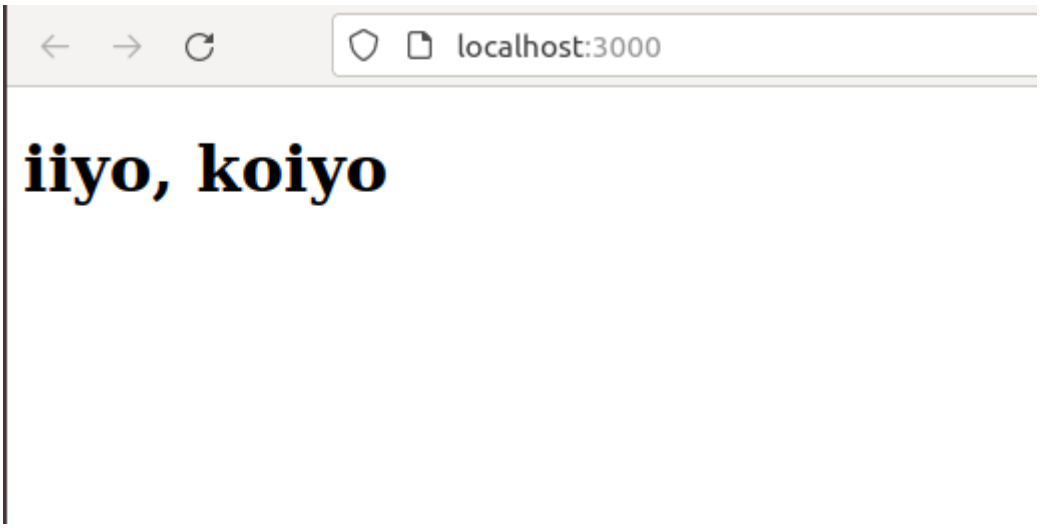
delete the files in src folder, and:

```
JS index.js M ×
src > JS index.js > ...
1   import React from 'react';
2   import ReactDOM from 'react-dom';
3
4   const element = <h1>iiyo, koiyo</h1>;
5   console.log(element);
```

react app:




```
1   import React from 'react';
2   import ReactDOM from 'react-dom';
3
4   const element = <h1>iiyo, koiyo</h1>;
5   ReactDOM.render(element, document.getElementById('root'));
6
7
```



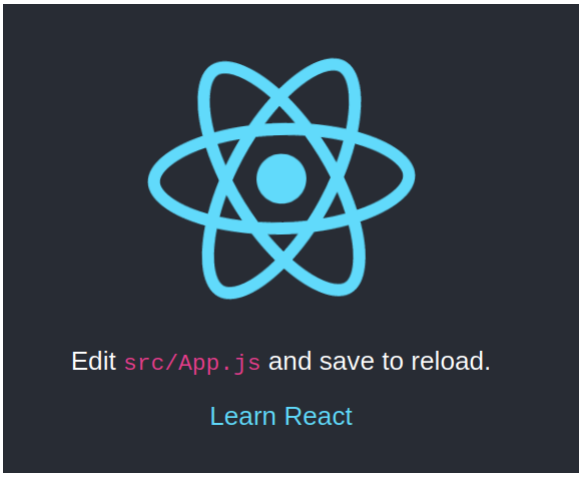
build a shopping cart website:

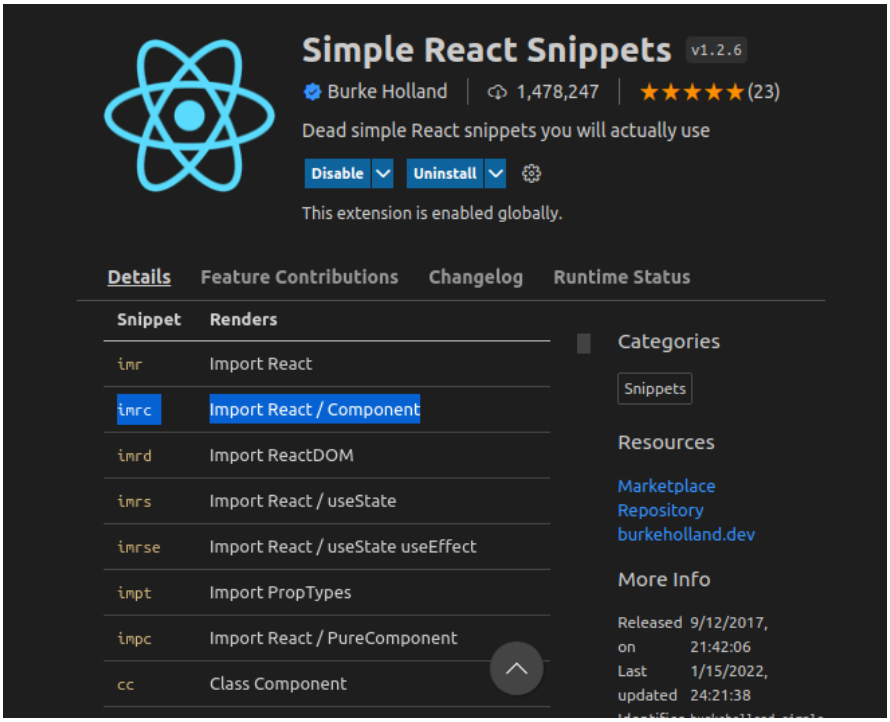
install bootstrap:

```
npm i bootstrap@4.1.1
```

then

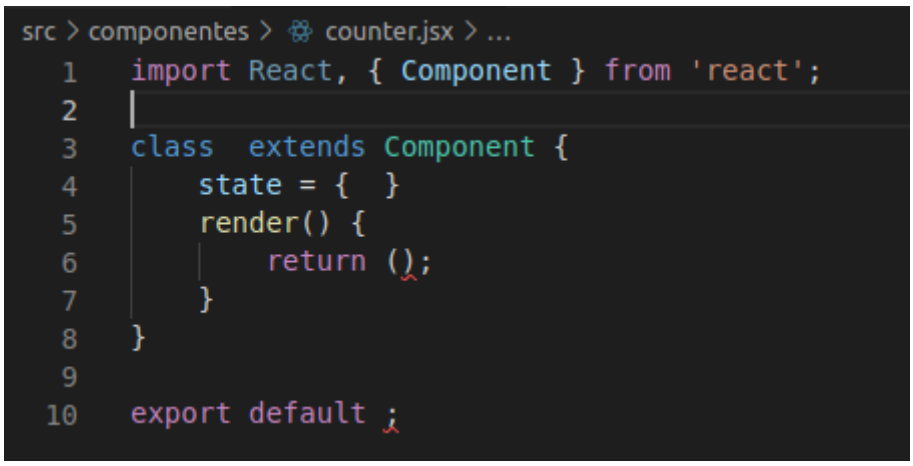
```
import 'bootstrap/dist/css/bootstrap.css';
```





imrc: shortcut for import React, { Component } from 'react';

cc:create class



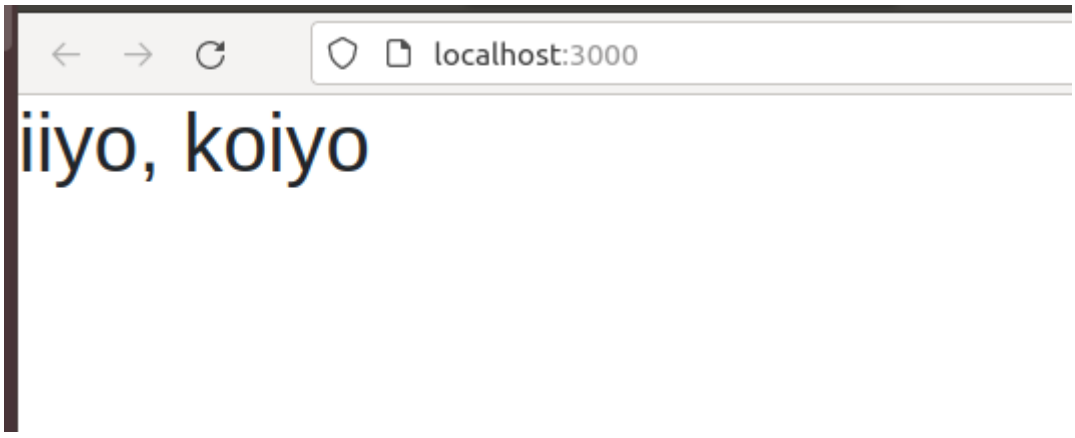
add class name Counter, delete state, and export default Counter;

```jsx
import React, { Component } from 'react';

class Counter extends Component {
    //state = {  }
    render()
    {
        return <h1>iiyo, koiyo</h1>;
    }
}

export default Counter;
```

we can also directly export class:

src > componentes > counter.jsx > Counter

```jsx
import React, { Component } from 'react';

export default class Counter extends Component {
    //state = {  }
    render()
    {
        return <h1>iiyo, koiyo</h1>;
    }
}
```



wrap text and button: two methods

index.js M | counter.jsx U

src > componentes > counter.jsx > Counter > render

```jsx
import React, { Component } from 'react';

export default class Counter extends Component {
    //state = {  }
    render()
    {
        //return <div><h2>iiyo, koiyo</h2><button>Increment</button></div>;
        return(
            <div>
                <h2>iiyo, koiyo</h2>
                <button>Increment</button>
            </div>
        );
    }
}
```

add state:

```
3   export default class Counter extends Component {
4       state = {
5           count:0
6       }; //data this Counter needs
7       render()
8       {
9           //return <div><h2>iiyo, koiyo</h2><button>Increment</button></di
10          return(
11              <React.Fragment>
12                  <span>{this.state.count}</span>
13                  <button>Increment</button>
14              </React.Fragment>
15          );
16      }
17      formatCount()
18      {
19          return this.state.count === 0? 'zero' : this.state.count;
20      }
21  }
```

or write a function:

```
17      formatCount()
18      {
19          const{count} = this.state
20          return count === 0? 'zero' : count;
21      }
22  }
```

we can return h1:

```
formatCount()
{
    const{count} = this.state
    //return count === 0? 'zero' : count;
    return count === 0? <h1>zero</h1> : count;
}
```

add button, style

```
styles={
    fontSize: 100,
    fontWeight: 'bold'
}
```

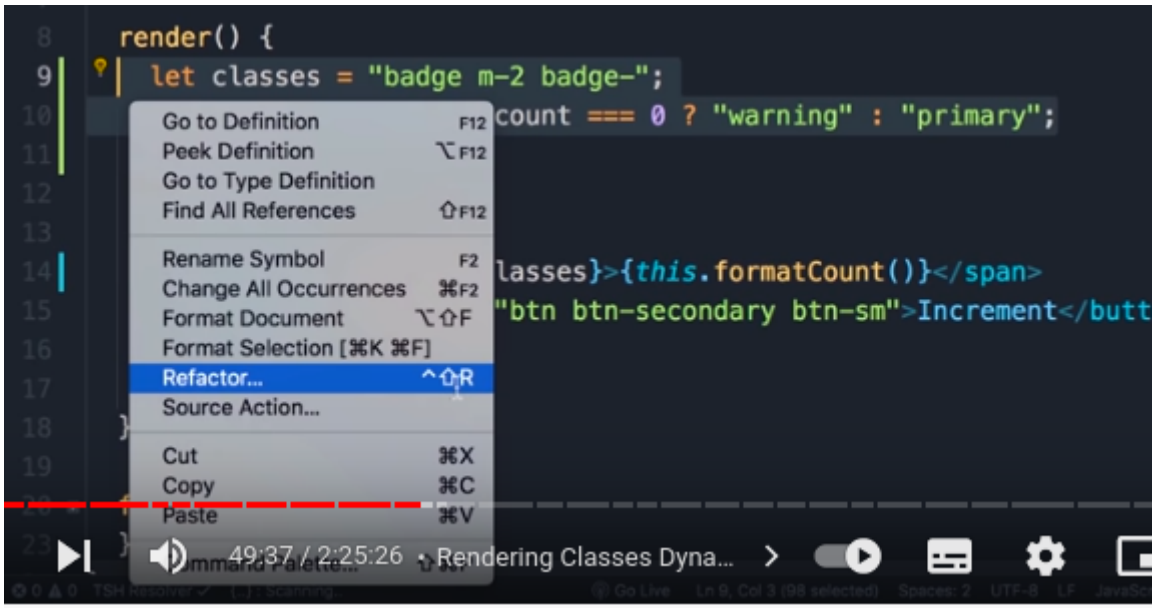
```
<React.Fragment>
    <span style={this.styles} className="badge badge-primary m-2">{this.formatCount()}</span>
    <button className="btn btn-secondary btn-sm">Increment</button>
</React.Fragment>
```

edit it inline:

```
<span style={{fontSize:50}} cl
<button className="btn btn-sec
```

edit classes, if count == 0 yellow, else blue

encapsulate the two lines: Refractor(ctrl+shift+R)



automatically generate:

```
newMethod() {
    let classes = "badge m-2 badge-";
    classes += (this.state.count === 0) ? "warning" : "primary";
    return classes;
}
```

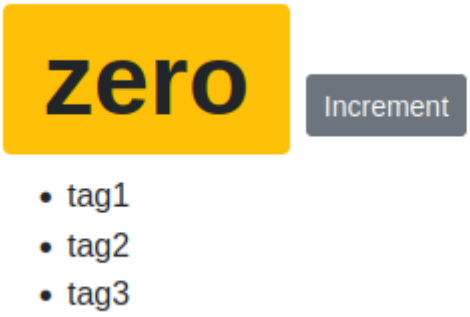
```
render()
{
    let classes = this.newMethod();
    //return <div><h2>iiyo, koiyo</h2><button>Increment</button></div>;
    return(
        <React.Fragment>
            <span style={{fontSize:50}} className={classes}>{this.formatCount()}</span>
            <button className="btn btn-secondary btn-sm">Increment</button>
        </React.Fragment>
    );//m-2:margin is 2
}
newMethod() {
    let classes = "badge m-2 badge-";
    classes += (this.state.count === 0) ? "warning" : "primary";
    return classes;
}
```

Render a list of items

jsx doesn't have loop(angular has)

create a list: use map to map a string to a list

```
<ul>{this.state.tags.map(tag => <li>{tag+"114514"}</li>)}</ul>
```



conditional rendering

jsx doesn't have if else

use js, if the tag list is empty, the website shows "there are no tags", else shows the tags

```
renderTags()
{
    if (this.state.tags.length == 0) return <p>there are no tags</p>;
    return <ul>{this.state.tags.map(tag => <li>{tag+"114514"}</li>)}</ul>;
}
```

js &&:

true && "a string"

print "a string"

true && "a string" && 1

print 1

a non-empty string is considered true, and if the whole sentence is true, js prints the last one

if the tags list is none, indicate people to add tags:

`{this.state.tags.length === 0 && "please create a new tag"}`

handle events

an edit trick: when editing and in pair, choose the blabla and ctrl+D, we can edit the pair on a single time

react onclick: **no brackets**, and the **C** is upper case

js:

```
</script>
<img loading="lazy" id="yajuusenpai" onclick="changeImage()" src="ikuiku.jpeg" width="200"
height="200">
```

react:

```
<button
    onClick={this.handleIncrement}
    className="btn btn-secondary btn-sm">
    Increment
</button>
```

the functions we wrote cannot use **this**:

```
handleIncrement()
{
    console.log("114514", this.state);
}
```

```
Uncaught TypeError: this is undefined
    handleIncrement          counter.jsx:10
    React 12
    attemptToDispatchEvent   bundle.js:13611
    dispatchEvent            React
    unstable_runWithPriority scheduler.development.js:456
    runWithPriority$1        bundle.js:18909
    React 3
[Learn More]
```

if `console.log("114514", this);`:

```
114514 undefined
```

bind the method to constructor

```
constructor()
{
    super();
    this.handleIncrement.bind(this);
}
```

"this" is always referencing the current object

(why should we use this "constructor"?)

then we can **reset** the handleIncrement function (replace the original handleIncrement with the binded function):

```
constructor()
{
    super();
    this.handleIncrement = this.handleIncrement.bind(this);
}

handleIncrement()
{
    console.log("114514", this.state.count);
}
```

now we can use sth in "this":

```
114514 0
```

or using an arrow function:

```
handleIncrement=()=>
{
    //this.state.count += 1;
    console.log("114514", this.state.count);
}
```

setState:

react cannot automatically renew the states

we should write this.setState() to renew it

```
handleIncrement=()=>
{
    this.state.count += 1;
    this.setState();
    console.log(this.state.count);
}
```

and the console shows 1,2,3,4.....

if we write this:

```
handleIncrement=()=>
{
    this.setState({count: this.state.count + 1});
}
```

we can add 1 to this.state.count and set the added number to this.state.count

Passing Event Arguments

we cannot pass some parameters to the handleIncrement function because we can only put function name in the onClick, but not function(parameters):

```
<button
    onClick={this.doHandleIncrement}
    className="btn btn-secondary btn-sm">
    Increment
</button>
```

so we write this function:

```
doHandleIncrement=()=>
{
    this.handleIncrement({id:1});
    //deliver parameters to handleIncrement
}
```

and:

```
handleIncrement=(product)=>
{
    console.log(product)
    this.setState({count: this.state.count + 1});
}
```

or inline function:

```
onClick={()=>this.handleIncrement(parameters)}
```

JSX

tree components

e.g. "counters" consist of several "counter"s

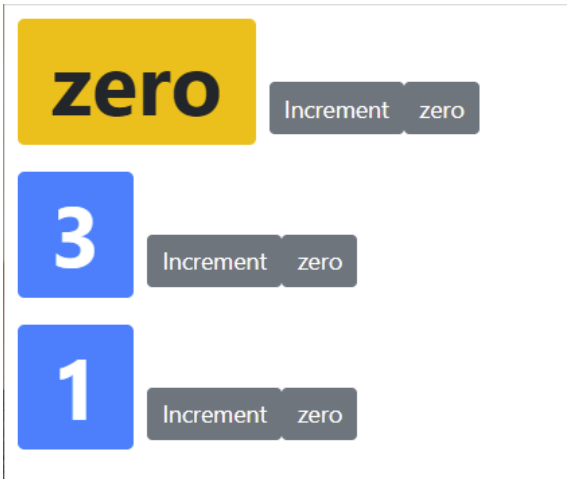
create a file "counters.jsx" under components:

like "counter", imrc, cc+tab

```
import React, { Component } from 'react';
import Counter from './counter';

class Counters extends Component {
    state = {  }
    render() {
        return (
        <div>
            <Counter />
            <Counter />
            <Counter />
        </div>
        );
    }
}
export default Counters;
```

and in index.js:

```
import Counters from './components/counters'

ReactDOM.render(<Counters/>, document.getElementById("root"));
```



set state in Counters:

```
state = {
        counters:[
            {id:1, value:1},
            {id:2, value:2},
            {id:3, value:0},
            {id:4, value:0}
        ]
      }
```

initialize the counters: use props

(what is props??????)

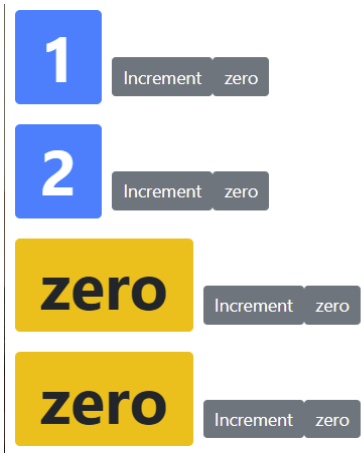
use map instead of writing counter for four times

```
<div>
    {this.state.counters.map(counter=><Counter
                                        key={counter.id}
                                        value={counter.value}
                                        selected={true}
                                        />)}
</div>
```

if we write `console.log('props', this.props)` in counter.jsx, we can get:

```
props ▶ {value: 1, selected: true}          counter.jsx:20
props ▶ {value: 2, selected: true}          counter.jsx:20
props ▶ {value: 0, selected: true}          counter.jsx:20
props ▶ {value: 0, selected: true}          counter.jsx:20
>
```

so we can use props.value to initialize count: `state = {count:this.props.value};`



Children

if we write sth in Counters, like h4:

```
<div>
    {this.state.counters.map(counter=>(
        <Counter key={counter.id} value={counter.value}>
            <h4>iiyo, koiyo</h4>
        </Counter>
    ))}
</div>
```

and in Counter render(): `console.log('props', this.props)`

```
props ▶{value: 1, children: {…}}        counter.jsx:20
props ▶{value: 2, children: {…}}        counter.jsx:20
props ▶{value: 0, children: {…}}        counter.jsx:20
props ▶{value: 0, children: {…}}        counter.jsx:20
>
```

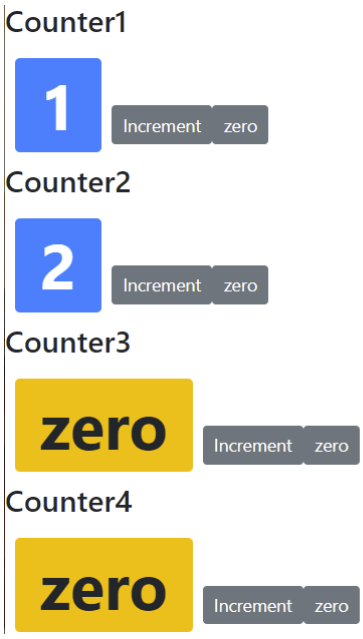
this h4 is the children of Counter

we can use props.children to change the title of each counter:

```
render()
{
    console.log('props', this.props)
    let classes = this.newMethod();
    return(
        <div>
            {this.props.children}
            <span style={{fontSize:50}} className={classes}>
            <button
                onClick={()=>this.handleIncrement({id:1})}
                className="btn btn-secondary btn-sm">
                Increment
            </button>
            <button
                onClick={this.returnToZero}
                className="btn btn-secondary btn-sm">
                zero
            </button>
        </div>
    );//m-2:margin is 2
}
```

and we can change the title dynamically

```
<h4>Counter{counter.id}</h4>
```



注:

```
render() {
    return (
    <div>
        {this.state.counters.map(counter=>(
        <Counter key={counter.id} value={counter.value}>
            <h4>Counter{counter.id}</h4>
        </Counter>
        ))}
    </div>
    );
```

这段的意思应该是（用类似python的伪代码写了一下）：

answer = []

for counter in this.state.counters:

    answer += Counter(key=xxx, props={value:xxx, children:那个h4})

return answer

或者更确切的应该是：

[Counter(counter.blabla) for counter in counters]

js的map应该跟python的[func(i) for i in sth_iterable]差不多

filter跟[i for in sth_iterable if func(i)]

states are **private**, internal, local

props are **public**, **read only**

Counter handle the states in Counters:

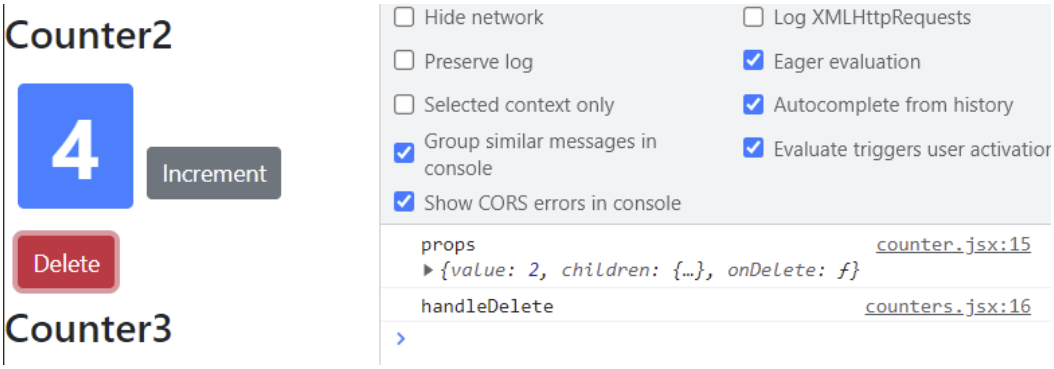
we should use props to transfer a function of Counters class

```
handleDelete=()=>
{
    console.log("handleDelete")
    //const counters = this.state.counters.filter(c=>c.id !== counterID)
    //this.setState({counters:counters});
};
render() {
    return (
    <div>
        {this.state.counters.map(counter=>(
        <Counter key={counter.id} onDelete={this.handleDelete} value={counter.value}>
            <h4>Counter{counter.id}</h4>
        </Counter>
        ))}
    </div>
    );
}
```

so we can use this.props.onDelete in counter.jsx:

```
<button
    onClick={this.props.onDelete}
    className="btn btn-danger btn-sm m-2">
    Delete
</button>
```

then we click the Delete button:



updating the state

we can adopt a parameter to the function handleDelete() and use it to update the

state of Counters:

```
handleDelete=(counterID)=>
{
    console.log("handleDelete, counterID is ", counterID)
    const counters = this.state.counters.filter(c=>c.id !== counterID)
    //上面那个应该是把id不等于counterID的滤掉了
    //类python伪代码[c for c in this.state.counters if c.id != counterID]
    //点击第二个delete按钮的时候，传进来的counterID=2
    //所以新的counters只有id为1,3,4的三个,把旧的counters替换掉
    this.setState({counters});
};
```

```
<button
    onClick={()=>this.props.onDelete(this.props.id)}
    className="btn btn-danger btn-sm m-2">
    Delete
</button>
```

怎么从父传给子，子传给父的。。。乱七八糟

we may write new things of each Counter, and rewrite the props is not easy, like sth below:

```
state = {
    counters:[
        {id:1, value:1, sth: 114514},
        {id......}
};

render() {
    return (
        <div>
            {this.state.counters.map(counter=>(
                <Counter
                    key={counter.id}
                    onDelete={this.handleDelete}
                    value={counter.value}
                    id={counter.id}
                    sth={counter:sth}>
                    <h4>Counter{counter.id}</h4>
```

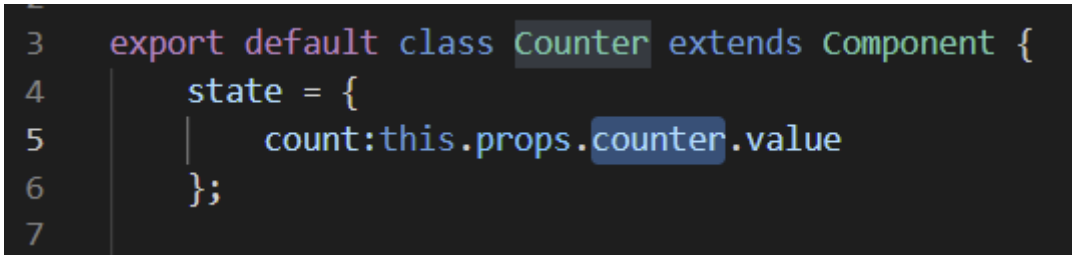
```
                </Counter>
            ))}
        </div>
    );
}
```

so we can give the whole counter: counter={counter}

```
{this.state.counters.map(counter=>(
    <Counter
        key={counter.id}
        onDelete={this.handleDelete}
        counter={counter}>
        <h4>Counter{counter.id}</h4>
    </Counter>
))}
```

and **change** the "this.props.id" to "this.props.counter.id"

note: be careful to find all the "this.props.blabla" and change it!!!!!



Removing Local State

controlled component

remove the state in counter.jsx and use this.props.counter.sth

multiple components and lift up

(I saved the code at this time)

copy the functions of counters.jsx to app.js, and use props to visit the functions

so the states of counters can be used by navbar

stateless functional component

for simple **stateless** components

```
const NavBar=(props)=>
{
    return (
        <nav class="navbar navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">
                Navbar
```

```
                <span className="badge badge-pill badge-secondary">
                    {props.totalCounters}
                </span>
            </a>
        </div>
        </nav>
    );
}
```

or use ({totalCounters}) and only give totalCounters from props:

```
const NavBar=({totalCounters})=>
{
    return (
        <nav class="navbar navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">
                Navbar
                <span className="badge badge-pill badge-secondary">
                    {totalCounters}
                </span>
            </a>
        </div>
        </nav>
    );
}
```

use props as parameter to give properties to the function

e.g.

```
<NavBar totalCounters={this.state.counters.filter(c=>c.value>0).length}/>
```

Destructuring

(not destruct)

```
const {onReset, onIncrement, onDelete, counters} = this.props;
```

use this to avoid writing many this.props

e.g. change `this.props.counter.id` to `counter.id`

Lifecycle Hooks

Mount: constructor, render, componentDidMount

Update: render, componentDidUpdate

Unmount: componentWillUnmount

only in components, not stateless functional component

```
APP constructor                                  App.js:20
App rendered                                     App.js:59
Navbar rendered                                  navbar.jsx:7
counters rendered                                counters.jsx:6
4 counter undefined rendered                     counter.jsx:15
APP mounted                                      App.js:24
>
```

use className instead of class in jsx!!!!!!

```
5  const NavBar=({totalCounters})=>
6  {
7      console.log("Navbar rendered");
8      return (
9          <nav className="navbar navbar-light bg-light">
10         <div className="container-fluid">
11             <a className="navbar-brand" href="#">
12                 Navbar
13                 <span className="badge badge-pill badge-secc
14                     {totalCounters}
15                 </span>
```