箭头函数

使用箭头（=>）来定义函数，大大减少代码量：

```
let fn = (a,b) =>a+b;
//等同于
let fn = function(a,b){
    return a+b;
}
```

- 大括号的解释问题 函数体代码多了就用他，显示写法，像这样：

```
let fn3 = (a,b)=>{
    a = a + b;
    console.log('ss');
}
```

当返回是个大括号对象，再箭头右侧时会被解释为代码块时需要再对象外面加上括号，像这样：

```
let fn3 = (a,b)=>({userName:userName,passWord:passWord})
```

- 箭头函数的this

1. 箭头函数中的this为定义时所在的对象，非使用时所在的对象。比如：

```
function fn() {
    let id = 2;
    return () => {
        return () =>{
            return () =>{
                return this.id
            }
        }
    }
}
```

上述内层函数多个return ,但都采用了箭头函数的形式，因此this均为定义时所再的

fn这个this

2. 不可进行this的指向改变 因为箭头函数本身没有自己的this，也就别指望通过call()/apply()/bind()等方法去改变this了。
3. for循环绑定事件 绑定事件中循环为每个相关元素事件，使用ES6的let时,使用箭头函数就不要使用this，此时的this为时间函数定义时的window全局（node则为global）。

```
ele.onClick = function (){
     for(let i = 0;i<xx.length;i++){
         this.className = '我是这个元素';//this为ele
     }
 }
 ele.onClick =>{
     for(let i = 0;i<xx.length;i++){
         this.className = '我是这个元素';//this为window
 }
```

- 不可作为构造函数 即父类不能采用箭头函数作为constructor,浏览器会友善的告诉你，erro~~
- 无可用arguments 可以采用...arr，即rest参数（获取多余参数，放入数组）

```
function fn(){
    setTimeout(()=>{
        console.log(arguments);
    },100);
}
//传参给fn
fn(1.2.3);
```

上述最后打印出的arguments为fn 的arguments,

- 箭头套箭头 上述也有，ES6中的箭头函数，可以嵌套的，同时也可使得高阶函数写法简洁化

案例

就像是数字、字符串、布尔值一样，函数也是值，意味着可以像传递其他数据一样传递函数，可以将函数作为参数传递给另外一个函数。

```
let multiply = (x) => (y) => x * y
multiply(5)(20)
```

高阶函数的定义是，接受函数作为参数的函数。 使用数组 map方法，它是一个数组遍历的方法，接受一个函数作为参数应用到数组中的每个元素。例如，可以像这样对一个数组作平方：

```
let square = (x) => x * x
[1, 2, 3].map(square)
```

为了便于理解，我们创建一个类似的map函数

```
let map = (fn, array) => {
  const mappedArray = []

  for (let i = 0; i < array.length; i++) {
    mappedArray.push(
      // apply fn with the current element of the array
      fn(array[i])
    )
  }

  return mappedArray
}
```

更简单的实现map = array => fn => Array.prototype.map.call(array, fn)

在组件化框架流行的今天，这种写法更是提高代码可读性的良好方法，这里我们小用下react的jsx语法

```
let yell = (Component) =>
  ({ children, ...props }) =>
    <PassedComponent {...props}>
      {children.toUpperCase()}!
    </PassedComponent>

let Title = (props) => <h1>{props.children}</h1>
let AngryTitle = yell(Title)

<UpperTitle>Whatever</UpperTitle>
```

最后这里输出 <h1>WHATEVER!</h1>