一、为什么学习函数式编程：

笔记本：	1.有关印象笔记		
创建时间：	2020/6/14 15:24	更新时间：	2020/7/19 10:19
作者：	1639079350@qq.com		
标签：	1.函数式编程范式, vue		

一、为什么学习函数式编程：

- vue3开始拥抱函数式编程；
- react的高阶组件用到了函数式编程，redux运用了函数式编程思想；
- 可以抛弃this；
- 打包过程中可以更好利用tree shaking过滤无用代码；
- 函数达到最高程度的复用；
- 有很多库可以帮助我们进行函数式开发：ladash/underscore/ramda
- 方便测试、方便并行处理；

函数式编程（FP）：
1.一种编程范式。【还有面向过程编程、面向对象编程】
2.思维：对运算过程进行抽象。用来描述数据（函数）之间的映射。
3.相同的输入会得到相同的输出。

```
// 非函数式
let num1 = 1;
let num2 = 2;
let numSum = num1 + num2;

// 函数式
function sum(a,b) {
    return a + b;
}
let numSum = sum(1,2);
```

函数是一等公民：在js中函数就是一个普通的对象，可以把函数存储到变量/数组中，它还可以作为另一个函数的参数和返回值，甚至我们可以在程序运行时通过new Function()来构造新函数。
==》函数可以存储在变量中；函数作为参数；函数作为返回值；

高阶函数：
1.可以把函数作为参数传递给另一个函数；

2.意义：抽象通用的问题，屏蔽细节，只关注目标结果；（函数作为参数使得函数更灵活，定义时不需要考虑函数的具体操作）
3.常用高阶函数：forEach、map 、filter、every、some、find、findIndex、reduce、sort
2.可以把函数作为另一个函数的返回结果；【没有返回值】

```
// 函数作为参数
        function forEach(arr, fn) {
            for (let i = 0; i < arr.lenght; i++) {
                fn(arr[i]);
            }
        }

        let array = [1, 2, 3, 4, 5];
        forEach(array, function (item) {
            console.log(item); // ------------------没有返回值
        })

        function filter(arr, fn) {
            let results = [];
            for (let i = 0; i < arr.lenght; i++) {
                if (fn(arr[i])) {
                    results.push(arr[i]);
                }
            }
            return results;
        }
        // 测试
        let arr = [1, 3, 4, 7, 8];
        let r = filter(arr, function (item) {
            return item % 2 === 0;
        })
        console.log(r); // 永远为[]
```

函数作为返回值：即一个函数生成另一个函数。【apply传参，只会调用一次】

```
        function once(fn) {
            let done = false;
            return function() {
                if(!done) {
                    done = true;
                    return fn.apply(this, arguments);
                }
            }
        }
```

```
let pay = once( function(money) {
    console.log(`支付: ${money} RMB`);
})
//  只调用一次
pay(5);
pay(5);
pay(5);
pay(5);
```

常用高阶函数:

```
// map
const map = (arr, fn) => {
    let results = [];
    for (let value of arr) {
        results.push(fn(value))
    }
    return results;
}
let arr = [1, 2, 3, 4];
console.log(map(arr, v => v * v)); // [1, 4, 9, 16]

// every
const every = (arr, fn) => {
    let result = true;
    for (let value of arr) {
        result = fn(value);
        if (!result) {
            break;
        }
    }
    return result;
}
let arr = [9, 12, 14];
console.log(every(arr, v => v > 10)); // false

// some
const some = (arr, fn) => {
    let result = false;
    for (let value of arr) {
        result = fn(value);
        if (result) {
            break;
        }
    }
    return result;
```

```
        }
        let arr = [1, 3, 4, 9];
        console.log(some(arr, v => v % 2 === 0)); // true
```