编程中的数学思想

函数式编程

LEE 2020.6.17

纯函数

定义：对于相同的输入，得到的输出结果相同，也就是数学意义上的函数。

```
function test(num) {
    return num + 1
}
```

```
function test(num) {
    return num ++
}
```

纯函数没有副作用，不改变任何的外部状态。JavaScript参数是按照引用传递的，简单的等号赋值，依然使用同一份引用

高阶函数

把函数当做普通的变量，既可以作为参数传递，也可以作为值返回。

['1', '2', '3'].map(ele => Number(ele))

['1', '2', '3'].map(ele => parseInt(ele));

map、every、some、filter、reduce等都是高阶函数

```
function test() {
    function A() {}
    return A;
}
```

函数式编程

- 函数式编程是一种编程范式，类似于命令式、声明式
- 函数式编程和语言无关，可以操作函数的语言都可以进行函数式编程
- 函数式编程是面向数学的抽象，函数的处理逻辑描述为表达式求值的过程
- 函数式编程中的变量只是求值中的过程量，是不可变的，不能够给一个变量多次赋值
- 函数和普通的数据类型一样，可以存储在对象，赋值给变量等等
- 函数不依赖外部的状态，也不产生新的状态，函数的结果只依赖于输入

Bad 栗子

```
[1, 2, 3].map(ele => Number(ele))
```

map函数的第一个参数是回调函数，回调函数的第一个参数是当前遍历到的数组项。

函数式的写法?

```
[1, 2, 3].map(Number)
```

函数之间相互包裹，并没有进行其他的操作，导致代码的可维护性降低

一道面试题

[1, 2, 3].map(parseInt)

parseInt(string, radix) 函数的第二个参数是解析string时的进制基数

radix是2-36之间的整数，radix值为0时，解析为10进制，无法解析的值返回NaN

parseInt(1, 0) =>

parseInt(2, 1) =>

parseInt(3, 2) =>

函数式之柯里化

值传递给函数一部分参数，让函数返回一个函数，去处理剩余的参数，把一个多参数的函数分解转化为单参函数。

```
function A(a, b) {
    return a + b
}
```

普通调用 => A(1, 2)

假设一个函数curry可以把普通函数转化为科里化函数

```
var B = curry(A)
B(1)
B(1)(2)
```

柯里化有什么用

```
function ajax(type, url, data) {
    var xhr = new XMLHttpRequest();
    xhr.open(type, url, true);
    xhr.send(data);
}
ajax('POST',  'www.test', 'a=1');    重复调用的时候参数冗余
var ajaxCurry = curry(ajax);
var post = ajax('POST');
var env1 = post('www.test');    复用性更高
```

柯里化的实现

```
var curry = function (fn) {
    var args = [].slice.call(arguments, 1);
    return function() {
        var newArgs = args.concat([].slice.call(arguments));
        return fn.apply(this, newArgs);
    };
};
function add(a, b) => a + b
var addCurry = curry(add)
addCurry(1,  2)
```

柯里化之函数组合

```
var toUpperCase = function(x) { return x.toUpperCase(); }


var hello = function(x) { return 'HELLO, ' + x; };

var greet = function(x){

    return hello(toUpperCase(x));

};

greet(‘kevin'); => 'HELLO, KEVIN'
```

简单的函数组合实现

```
var compose = function(f,g) {
    return function(x) {
        return f(g(x));
    };
};
var greet = compose(hello, toUpperCase);
greet(‘kevin');
```

利用 compose 将两个函数组合成一个函数，让代码从右向左运行，而不是由内而外运行，可读性大大提升。这便是函数组合。

工具库

- loadsh.js
- underscore.js
- [ramda.js](https://ramda.cn/)

没有银弹，没有答案，不需要刻意强求