一、compose函数

什么是compose函数?

- 将需要嵌套执行的函数平铺
- 嵌套执行指的是，一个函数的返回值将作为另一个函数的参数

所谓的compose，在函数式编程中，就是将几个有特点的函数拼凑在一起，让他们结合，产生一个崭新的函数。

compose函数的作用?

实现函数式编程中的PointFree，使我们专注于转换而不是数据。

也就是说，我们完全可以把数据处理的过程定义成一种与参数无关的合成运算，不需要用到代表数据的那个参数，只要把一些简单的运算步骤合成在一起即可，而PointFree指的是不使用所要处理的值，只合成运算过程，即我们所指的无参数分隔。

现在我们有一个这样的需求，计算一个数加上10再乘以10的值

很多人会这么做

```
let calculate = x => (x + 10) * 10
console.log(calculate(10)); // 200
```

这样写是没有问题的，但是这样写是典型的命令式编程，这样的代码完全不具备复用性。作为一个函数式编程开发者，会对这段代码进行抽象来实现简化代码的目的，函数式编程就是将我们的程序分解为一些更可重用、更可靠且更易于理解的部分，然后再将它们组合起来，形成一个更利于推理的程序整体。

以上面的需求为例，我们关注的是它的动作，先加上10，再乘以10，改写如下

```
// 将动作分解
let add = x => x + 10;
let mul = y => y * 10;

// 再将add函数的计算结果作为mul函数的参数
console.log(mul(add(10))); // 200
```

这样也能实现我们的需求，但是根据复合函数的定义，我们需要将上述代表两个动作的两个函数合成一个函数。

分析两个函数的特性

1. 两个函数都有一个共同的参数
2. 函数的执行顺序是从右到左
3. 前面函数执行的结果交由后面的函数处理

根据特性，使用闭包实现如下代码：

```
// 将动作分解
let add = x => x + 10;
let mul = y => y * 10;

// 创建compose函数
let compose = (f, g) => {
  return function (x) {
    return f(g(x));
  }
}

// 执行顺序从右往左即先执行add再执行mul
let calculate = compose(mul, add);
console.log(calculate(10)); // 200
```

这样的compose没有通用性，因为它只能处理两个函数，接下来我们实现一个通用的compose函数。

通用的compose函数有如下几个步骤：

1. 将传入的函数当做参数收集起来
2. 将上一次函数的执行结果变成参数传递给下一个函数，这里我们可以使用reduce实现，需要注意的是复合函数的执行顺序是从右往左执行，而reduce函数的执行顺序是从左往右执行，所以这里使用reduceRight进行实现。

```
// 将动作分解
let add = x => x + 10;
let mul = y => y * 10;
let sub = z => z - 10;

// 创建compose函数
let compose = function () {
  // 将参数收集起来并将类数组转为数组对象
  let args = [].slice.call(arguments);
  return function (x) {
    // 返回执行结果
    return args.reduceRight(function (res, cb) {
      return cb(res);
    }, x);
  }
}

let calculate = compose(mul, add);
let calculate2 = compose(sub, mul, add);
console.log(calculate(10)); // 200
console.log(calculate2(10)); // 190
```

更简单的写法是使用ES6

```
// es6写法
const compose = (...args) => x => args.reduceRight((res, cb) => cb(res), x);
```

我们通过compose函数在开发中可以设计和抽象功能到具体的函数里面，以便后期的复用，而且更多的时候compose函数的存在其实是服务于中间件的，redux中间件就是通过compose函数来实现的，webpack的loader加载顺序也是从右往左，这是因为webpack选择了compose方式实现。

二、pipe函数

什么是pipe函数?

pipe和compose都是函数组合，组合是函数式编程中非常重要的思想，就是将多个函数组合在一起以便能构建出一个新函数。在函数式编程中，纯函数应该被设计为只做一件事，如果想实现多个功能，可以通过函数的组合来实现。

pipe函数是compose函数的复制品，唯一修改的是数据流方向，它和compose函数所做的事情相同，只不过交换了数据流方向，也就是说compose函数的执行顺序是从右向左，最后一个函数会首先执行，之后传递结果给下一个函数执行，以此类推一直到最左侧的函数执行完，而pipe函数正好相反。

```
// 将动作分解
let add = x => x + 10;
let mul = y => y * 10;
let sub = z => z - 10;

let pipe = function () {
  let args = [].slice.call(arguments);
  return function (x) {
    return args.reduce(function (res, cb) {
      return cb(res)
    }, x);
  }
}

let a = pipe(add, mul);
let b = pipe(add, mul, sub);
console.log(a(10)); // 200
console.log(b(10)); // 190
```

```
60    // ES6写法
61    const pipe = (...args) => x => args.reduce((res, cb) => cb(res), x);
```