> 面向对象编程(OOP)通过封装变化使得代码更易理解。 函数式编程(FP)通过最小化变化使得代码更易理解。 -- Michacel Feathers（Twitter）

函数式编程

资源

1. 函数式编程指南
2. 阮一峰函数式编程入门教程

概念

函数式编程倡导**利用若干简单的执行单元**让计算结果不断渐进，逐层推导复杂的运算。

函数式编程有两个最基本的运算：合成（compose）和柯里化（Currying）。

合成（compose）

如果一个值要经过多个函数，才能变成另外一个值，就可以把所有中间步骤合并成一个函数，这叫做"函数的合成"（compose）。

合成的好处显而易见，它让代码变得简单而富有可读性，同时通过不同的组合方式，我们可以轻易组合出其他常用函数，让我们的代码更具表现力。

```
function f1(arg) {
  console.log("f1", arg);
  return arg;
}
function f2(arg) {
  console.log("f2", arg);
  return arg;
}
function f3(arg) {
  console.log("f3", arg);
  return arg;
}

function compose(...funcs) {
  if (funcs.length === 0) {
    return arg => arg;
  }
  if (funcs.length === 1) {
    return funcs[0];
  }
  return funcs.reduce((a, b) => (...args) => a(b(...args)));
}

let res = compose(f1, f2, f3)("omg"); //f1(f2(f3("omg")));

console.log("res", res); //sy-log
```

柯里化（Currying）

柯里化（英语：Currying），又译为**卡瑞化**或**加里化**，是把接受多个参数的函数变换成接受一个单一参数（最初函数的第一个参数）的函数，并且返回接受余下的参数而且返回结果的新函数的技术。

所谓"柯里化"，就是把一个多参数的函数，转化为单参数函数。

```
// 柯里化之前
function add(x, y) {
  return x + y;
}

add(1, 2) // 3

// 柯里化之后
function addX(y) {
  return function (x) {
    return x + y;
  };
}

addX(2)(1) // 3
```

这样调用上述函数：`(foo(3))(4)`，或直接 `foo(3)(4)`。

看下面的例子，这里我们定义了一个 `add` 函数，它接受一个参数并返回一个新的函数。调用 `add` 之后，返回的函数就通过闭包的方式记住了 `add` 的第一个参数。一次性地调用它实在是有点繁琐，好在我们可以使用一个特殊的 `curry` 帮助函数（helper function）使这类函数的定义和调用更加容易。

```
var add = function(x) {
  return function(y) {
    return x + y;
  };
};

var increment = add(1);
var addTen = add(10);

increment(2);
// 3

addTen(2);
// 12
```