5. 编程基础V：函数式编程范式

一. 避免重复.

二. 函数式编程范式.

1. 过程抽象（高阶函数）

· 一种抽象

· 过程作为参数

Example:

计算前100个整数的和、平方和、立方和.

2. 数据抽象（有序对、层次性数据、符号数据）

(1) 链表（list）

缩放表（scale list）

映射集（map）：返回将这一过程应用于表中各个元素得到的结果形成的表.

(2) 树（tree）

数值树（scale tree）：返回一棵具有相同形状的树，树中的每个数值都乘以这个因子.

3. 信号流（枚举器+过滤器+映射+累积器）.

(1) 枚举器 → 过滤器 → 映射 → 累积器

Example：计算叶结点上数值为奇数的平方和.

(2) 枚举器 → 映射 → 过滤器 → 累积器

Example：计算所有偶数的Fibonacci数列.

Example:

- 创建一个Messages Collection
- List<Message> messages = new ArrayList<>();
- messages.add(new Message("aglover", "foo", 56854));
- messages.add(new Message("aglover", "foo", 85));
- messages.add(new Message("aglover", "bar", 9999));
- messages.add(new Message("rsmith", "foo", 4564));

- 具体来讲，我希望找到Message当中所有延迟周期超过3000秒的条目并计算它们的总计延迟时长。

- Java 8 Stream API
  - long totWaitTime = messages.stream().filter(m -> m.delay > 3000).mapToLong(m -> m.delay).sum();
    枚举器为流 过滤器 映射 累加器

```
long totalWaitTime = 0;
for (Message message : messages)
{
    if (message.delay > 3000)
    {
        totalWaitTime += message.delay;
    }
```

4. 函数式编程特点：

(1) 函数被当作头等公民，意味着函数可以作为别的函数的参数、函数的返回值，赋值给变量或存储在数据结构中，通常以高阶函数的形式存在。

(2) 严格的函数式编程要求函数必须没有副作用，意味着影响函数返回值的唯一因素就是其参数。

(函数副作用：当调用函数时，除了返回值之外，还对外部作用域产生附加的影响，例如修改函数外的变量或修改参数。)

(3) 在函数式编程中，函数就是基础元素，可以完成几乎所有的操作，

A. 在函数式编程中，函数就是基础元素，可以完成几乎所有的操作，哪怕是最简单的计算，也是用函数来完成的，而我们平时在其他类型中所理解的变量（可修改，往往用来保存状态）在函数式编程中，是不可修改的，这意味着状态（State）不能保存在变量中，而事实上函数式编程是使用函数参数来保存状态，最好的例子便是递归。

三. 编程的现实考量.

1. 复杂性与规模.

2. 逻辑与物理.

3. 时间与空间.

(1) 算法驱动.（CPU计算时间）

(2) 数据驱动.（硬盘存储空间）

四. 证明程序的正确性.

- Edsger W. Dijkstra
  - Enumeration
  - Mathematical induction
  - Abstraction
- C. A. R. HOARE
  - Axioms proof