### 为什么

学习一种全新的编程范式，困难并不在于掌握新的语言。真正考验人的，是怎么学会用另一种方式去思考。例如，用“函数式”的方式编写代码牵涉到诸多方面，包括设计上的种种取舍，不同重用单元的作用等——比起语法，更看重思路。

1. 范式转变

传统编程风格（命令式的循环）和函数式特征更明显的方式 对比：

三个操作：将所有单词条目转换为小写；滤除虚词；计算余下单词的词频

1. 建立一个虚词的集合，实现目标方法; 1.lambda块方式实现的高阶函数——
2. 方法中首先建立一个map映射结构来 map, filter, forEach

容纳由单词和词频组成的键值对，接着 2.stream API

构造一个用来识别单词的正则表达式； 首先，建立一个列表来容纳正则表达式的

其次，逐一遍历所有找到的单词，转换 识别结果，并实现列表方法；

小写排除虚词情况，将首次遇到的单词 在方法里逐一遍历所有找到的单词，添加

添加map结构，将重复遇到的单词的 到列表结构中，并返回列表。

出现次数+1。 其次，实现目标方法；在方法里，先建立

***总体风格：以步进方式处理集合遍历*** 一个map映射；再将列表方法转换为 stream，以方便后续的相互独立的三项操 作——将所有单词条目转换为小写；滤除 虚词；计算余下单词的词频。

***总体思考方式——以Unix风格：***

***做完一步再做下一步，且每一步都相互独 立。***

**传统**

**风格**

**代码**

**交织**

**混乱**

**【**

**折中**

**】**

如果命令式编程语言也采用这种风格，必须对集合进行三次遍历——

将所有单词条目转换为小写；滤除虚词；计算余下单词的词频。但是这种写法性能效率过差。换句话说，就是牺牲了代码的清晰来换取执行性能。

**Unix风格 【理想风格】 函数式风格**

**性能较差 性能优化与代码简洁**

【注：

交织：穿插缠绕的合为一体，使错综复杂。命令式编程风格常常迫使我们处于性能考虑，把不同的任务交织起来，以便能够用一次循环来完成多个任务。

】

【注：

函数式编程使用map,filter等这些高阶函数将我们解放，使我们站在更高的抽象层次去思考问题，以破除交织现象。

】

1. 跟上语言发展的潮流 ——**所有的编程语言都日渐向函数式靠拢**

***比如***：Groovy已经具备了丰富的函数式特性，包括像“记忆”这样的高级特性在内。【注：记忆是指运行时自动缓存函数返回值的能力】

java 8也纳入了lambda块（也就是高阶函数），以提升函数式编程的能力。

Javascript语言本身就带有很多的函数式特性。

C++也在语言标准里增加了lambda 块，引人关注的就是Boost.Phoenix等类库。

***——只有学会这些新的编程范式，才能从容地利用它们。***

***比如***：函数式语言不仅在java虚拟机（JVM）平台上迅速展露头角——Scala和Clojure；

.NET平台也不例外—— F#已经是堂堂正正的一员。

1. 把控制权让渡给语言/运行时—— 虚拟机跨平台和自动垃圾收集等

条例：最好还是把时间花在**更高层次的抽象**上，多考虑怎样解决复杂的业务场景，少去费心复杂的底层操作。人生苦短，远离malloc；

当然，还有很多其他的乏味单调的任务，请将它们托付给语言本身和运行时。

更多地专注于更重要的问题。

函数式语言本身就是这样设计的——将琐碎的细节交付给运行时，令繁冗的**实现**华为轻巧。***比如***：使用高阶抽象取代基本的控制结构，统一为“数据结构+表达式”来处理。

1. 简洁

函数式抽象和面向对象抽象的关键区别：

函数式编程使用少数几个核心数据结构，围绕它们去建立一套充分优化的运转机构 。面向对象编程则不断地创建新的数据结构和附属的操作，建立新的类和类间的消息。

面向对象编程把所有的数据结构封装成类，一方面压制了方法层面的重用；另一方面则鼓励了大粒度的框架式的重用。函数式编程的程序构造更方便我们在比较细小的层面上重用代码。

面向对象编程通过封装不确定因素来使代码能被人理解。

1.封装，作用域，可见性等这些oop构造，其存在意义就是为了精细地控制‘谁’能感知状态和改变状态；但是当涉及多线程的时候，对状态的控制就相当复杂。上述那些机制就属于“不确定因素”。

—著名的<不确定性原理>如是说

2.在面向对象的命令式编程语言里面，重用的单元是 ：类和类之间沟通用的消息，这些都可以使用类图来表述。

3.oop的世界提倡开发者针对具体问题建立专门的数据结构，相关的专门操作以“方法”的形式附加在数据结构上。

比如：java世界的数十种XML类库，每一种都有自己定义的内部数据结构。

函数式编程通过尽量减少不确定因素来使代码被人理解。

1.与其建立种种机制来控制可变的状态，不如尽可能<消灭可变的状态>这个不确定因素。

——消除<变量/抽象/其他的不确定因素>

1. 函数式编程语言实现重用的思路很不一样。

函数式语言提倡*在有限的几种关键数据结构（list，set，map）*上运用针对这些数据结构*高度优化的操作*，以此构成*基本的运转机构*。开发者再根据*具体用途*，插入自己的数据结构和高阶函数去*调整*机构的运转方式。

1. 比起一味创建新的类结构体系，把封装的单元降低到函数级别，更有利于达到细粒度的，基础层面的重用。

比如：Clojure库中map方法封装了自动并行的能力，使得开发人员不用写一行代码就可以获得性能的大幅提升。