Atitit 惰性求值的优缺点

## 关于惰性求值

在函数式编程中，一般有三种结果输出函数式操作后的结果：

* 惰性求值：内部保存着最原始数据与处理函数，List被调用时才会进行执行函数，特点是循环效率最高，但是在进行RPC或者serialization时，结果不可预料，需要**保证元素被遍历**。每次写线上代码时，用惰性求值都感觉是一个定时炸弹。
* 修改内部值：内部数据在处理函数被遍历时被修改，特点是空间占用最小，但是某些数据结构下比如ArrayList进行filter操作时，时间复杂度爆高。
* 新拷贝：新建一个对象，并通过原始List与函数依次赋值。这个操作最简单安全，一般也在自己的工具类中这样写，但是执行效率较低，空间比较浪费。

//进行函数式变化时，需要新建一个对象

newList = new ArrayList<>();for(String s: llist){

newList.add("Hello " + s );

}

目前FP中，比如Haskell、Java8、Guava等函数式内部实现均采用了惰性求值。

### 如何安全地生成新拷贝对象

刚刚已经说过了，惰性求值过度优化可能导致奇葩结果。为了安全地传递给下家，可能需要将惰性求值转换为新值。

第一种方法(最推荐)，如果想偷懒的话，直接创建一个一定执行遍历操作的对象，将返回一个不可变的数组

FluentIterable.from(in)

.transform(new Function<Integer, String>() {

public String apply(Integer input) {

return Integer.toHexString(input);

}

}).toList();

或者下面这样的，但是更丑，结果比较保险，是常见的数组List

ImmutableList.copyOf(...)

Lists.newArrayList(...)

对比Java8的惰性求值，还是Java8更简洁

List<String> o = in.stream().map(new Function<Integer, String>() {

public String apply(Integer integer) {

return Integer.toHexString(integer);

}

}).collect(Collectors.<String>toList());

目前我已经很少使用Guava，主要是换成了Stream与Groovy，可以参考[这里](https://link.jianshu.com?t=https://miao1007.gitbooks.io/the-way-to-be-a-true-programmer/content/datastruct/streamapiyu-bi-shi-ti.html" \t "_blank)

作者：BlackSwift  
链接：https://www.jianshu.com/p/58529ca715a3  
來源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。