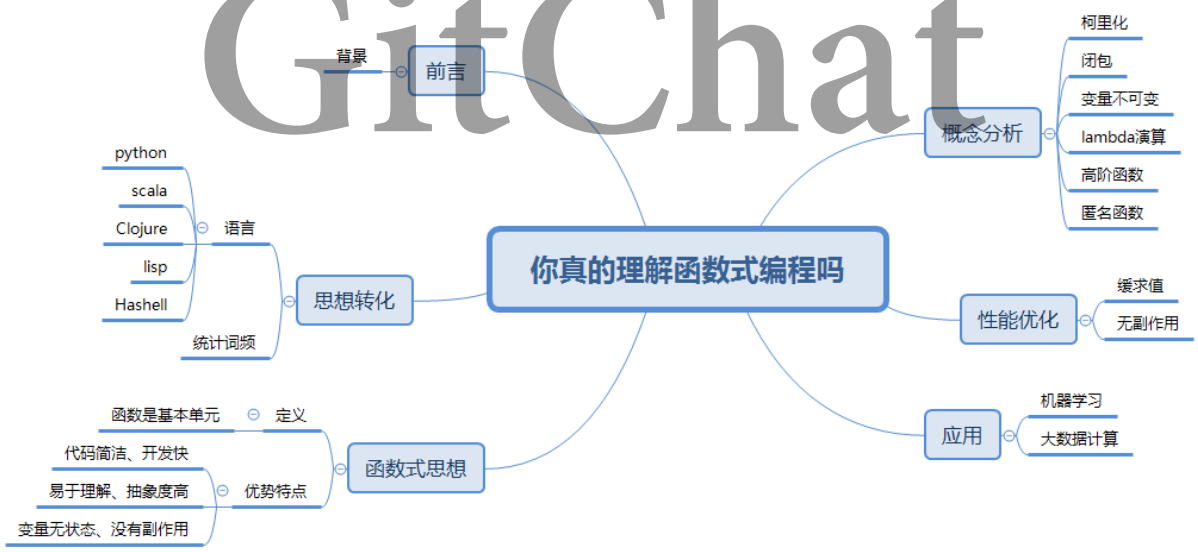
你真的理解函数式编程吗？

前言

现在机器学习、人工智能的发展趋势如火如荼，很多培训班也在引入大数据、机器学习的课程，受到众多IT从业者的追捧，有一种势必与传统模式一决高下，分享半壁江山之势，如果人工智能走向企业、社会，它带来的经济效益以及影响是巨大的，我们每个人不管从事什么行业都有必要了解一下人工智能的发展趋势，机器学习背后隐藏着什么不可告人的秘密，让我们来揭开层层面纱，一窥究竟。

在面向对象语言满天飞的现在，很多人认为，函数式编程是一种仅仅存在于某些偏门语言中，然而，纵观现代主流语言都在引进函数式特性，不同语言可能引进的程度不同，如我们最熟悉的jdk，jdk8已经引入了函数式编程的一些特性，可见大公司在不断引进新技术，唯有跟着技术前沿发展，我们才能立足于不败之地。

本篇文章的目录结构如下：



为什么函数式编程会在这个时期流行起来？这种思想出现至少10年以上了，那一定是它自身的某些特性符合了这个时代的特性，解决了企业中设计、开发中遇到的各种问题，这是一个流程技术的内在驱动力，在函数式编程里面函数的地位很高，如面向对象中的对象一样常见，有三个俗称三板斧的利器，map、filter、reduce，它将在以后我们的编程中随手不离，函数式编程为我们开启了另一个思考维度。

思想转化

计算机科学的进步经常是间歇式的发展，先形成好的思路、然后出来这种思路下面的语言，之后便会成为某一时代的主流，例如第一种面向对象的语言simula 1967年发明，直到1983年的面向对象的C++才流行起来，早年java总被认为太慢，编程迟钝，内存消耗太高不适合高性能的应用，如今计算机硬件已经不再是制约条件，硬件的变迁把它变为了极具吸引力的语言，从国内大部分公司应用即可看出来。

那么大家想一下，如今硬件仍然在飞速发展，那么更高级的面向函数式编程、语言会不会更适应这个时代？

函数式思想

什么是函数式编程？

百科定义：

> 函数式编程是种编程方式，它将电脑运算视为函数的计算。函数编程语言最重要的基础是λ演算（lambda calculus），而且λ演算的函数可以接受函数当作输入（参数）和输出（返回值）。

个人理解就是我们的编程是以函数作为单元来处理各个业务逻辑，函数既可以当做参数传来传去，也可以作为返回值，可以把函数理解一个值到另一个值得映射关系，由于函数式编程方式更适合于数据处理，随着存储器容量升高、计算机处理能力大幅提高，它的优势更加明显，最近支持函数式编程的语言也逐渐流行，比如python、scale等都因它们对函数式编程的支持被人们重视，从被遗忘的角落重新拾起。

函数式编程因为其特点更适用于统计分析数据、科学计算、大数据处理等方面工作，当然并不限于这些，在web开发、服务器脚本等其它方面也很不错，而面向对象编程更适合于开发和处理业务性强、功能模块完备的大型业务系统。

优势特点

代码简洁、开发快速

函数式代码同命令式相比代码量要少很多，一行顶十行，所以实现一些功能也比较简洁，作为开发者的我们，还是对代码更亲切一些，来看一个具体例子，从中体会其特点。功能描述：统计文本或网页中单词的频率TF（term frequency），词频在计算网页质量、信息检索中是一个重要概念，下面来看一下简化代码：

命令式实现

```
/**
 * Created by lilongsheng on 2017/11/5.
```

```
 */
public class Words {

    /**
     * 虚词过滤
     */
    private static Set<String> NON_WORDS = new HashSet<String>(){
        {add("the");add("and");add("of");add("to");add("a");
         add("i");add("it");add("in");add("or");add("is");add("d");
         add("s");add("as");add("so");add("but");add("be");}};

    /**
     * 命令式方式过滤实现
     * @param words
     * @return
     */
    public static Map wordFreq1(String words){
        TreeMap<String,Integer> wordMap = new TreeMap<>();
        Matcher m = Pattern.compile("\\w+").matcher(words);
        while (m.find()){
            String word = m.group().toLowerCase();
            if (! NON_WORDS.contains(word)){
                if (wordMap.get(word) == null){
                    wordMap.put(word,1);
                }else {
                    wordMap.put(word,wordMap.get(word)+1);
                }
            }
        }
        return wordMap;
    }
```


函数式实现

```
    /**
     * 将待处理对象转为列表集合
     * @param words
     * @param regex
     * @return
     */
    private static List<String> regexToList(String words,String regex)
{
        List wordList  = new ArrayList<>();
        Matcher m = Pattern.compile(regex).matcher(words);
        while (m.find())
            wordList.add(m.group());
        return wordList;
    }
```

```
    /**
     * 对集合统一处理
     * @param words
     * @return
     */
    public static Map wordFreq2(String words){
        TreeMap<String,Integer> wordMap = new TreeMap<>();
        regexToList(words,"\\w+").stream()
                .map(w -> w.toLowerCase())
                .filter(w -> !NON_WORDS.contains(w))
                .forEach(w ->
wordMap.put(w,wordMap.getOrDefault(w,0)+1));
        return wordMap;
    }

    public static void main(String[] args) {
        String test = "Before the popularity of network communicative
tools, such as Weibo and WeChat, people get used to know a person by
face and face talk, but now the situation has changed, the young
generation tend to know a person by network social communicative
tools. If they are interested in making friends, then they will be
friends online first and then keep trace with the former information
on the record. It seems that we can learn a person so fast and
convenient. There is a short video about a girl dated a guy, but the
guy did not have any account on the Internet, then the girl felt not
comfortable and started to question if the guy was a criminal. The
video satirizes people to rely on the Internet too much. They rather
to communicate with a person by the Internet instead of face and face
talk, while the latter is much trustworthy";

        System.out.println(wordFreq1(test).toString());

        System.out.println(wordFreq2(test).toString());
    }
}
```


函数式编程思维是对集合统一处理、统一操作，而命令式编程需要取出来每个单词单独处理，单独计数，而函数式只需要传入待处理对象集合、处理规则，我们不需要关注于具体细节，这样编程不仅仅减少了出现bug的概率而且提高了IT人员开发效率，何乐而不为呢。

易于理解，抽象度高

让我们再来看一个在开发中，我们经常遇到场景，例如我们有一个List列表，我们要把user的某个属性提取出来生成一个新的List，如下：

```
import lombok.Data;

/**
 * Created by lilongsheng on 2017/11/8.
```

```
 * 用户数据实体对象
 */
@Data
public class User {

    /**
     * 主键
     */
    private Integer id;
    /**
     * 用户名
     */
    private String userName;
    /**
     * 用户密码
     */
    private String userPassword;
    /**
     * 年龄
     */
    private Integer age;
    /**
     * 电话
     */
    private String phone;

    /*其它属性*/
}

public static void main(String[] args) {

        //初始化数据
        List<User> userList = new ArrayList<>();
        for (int i = 0; i < 10; i++) {
            User user = new User();
            user.setUserName("lilongsheng" + i);
            userList.add(user);
        }

        //命令式方式实现
        List<String> newUserNameList1 = new ArrayList<>();
        for (int i=0;i<userList.size();i++){
            User user = userList.get(i);
            newUserNameList1.add(user.getUserName());
        }


        //函数式方式实现 java 8 funcational
        List<String> newUserNameList2 = userList.stream()
                                                .map(p ->
p.getUserName())
```

```
    .collect(Collectors.toList());


        }
```

假设你已经了解了函数式语言的语法，你可能会觉得函数式写法很简洁，函数式编程并不需要你关注细节实现，我们在获取用户名作为一个新List时并没有对单独user对象操作，而是是告诉集合对象，我们要做什么，思维重心和关注点从“怎么做”转移到了“做什么”，通过map这个高阶函数把集合以及怎么做的规则传入，它帮我们处理集合元素，并返回一个结果集。

没有副作用，变量无状态

如果一个函数内外有依赖于外部变量或者环境时，常常我们称之为其有副作用，如果我们仅通过函数签名不打开内部代码检查并不能知道该函数在干什么，作为一个独立函数我们期望有明确的输入和输出，副作用是bug的发源地，作为程序员开发者应尽量少的开发有副作用的函数或方法，副作用也使得方法通用性下降不适合扩展和可重用性。

函数式编程语言强烈要求使用者编写没有副作用的函数，它的函数式数学意义上的函数，给定一个输入就会有一个输出，而且每次相同的输入输出也肯定一样，函数式中的变量所代表的意义并不是内存中的一块存储单元，因此多个函数同时操作一个变量或者一个函数调用多次一个变量都不会改变该变量的值，函数每次对于给定的输入都是作为一个新值来处理，就是因为它这种没有状态、没有副作用的理念，很适合大数据计算和处理，它只接受固定输入既可以得到预定计算结果，完全不依赖于外部环境，由这个想到了吕迅说过“我们要做一个纯粹的人、做一个大写的人”，不能因为环境、外部压力所屈服，以后，就让我们来编写无副作用的函数吧。

函数式编程其中的“函数”并不是我们计算机中理解的方法或函数，而是纯数学领域函数的概念，看一下百科中对数学函数的定义：

函数的定义：

> 给定一个数集A，对A施加对应法则f，记作f（A），得到另一数集B，也就是B=f（A）。那么这个关系式就叫函数关系式，简称函数。函数概念含有三个要素：定义域A、值域C和对应法则f。其中核心是对应法则f，它是函数关系的本质特征。

命令式程序是基于冯诺依曼计算机结构为基础，一条一条的执行操作指令，程序执行效率为每条命令执行的总和，我们是面向命令编程，一般我们使用的语言都是命令式语言，比如c c++ java 等等,还有一种编程方式叫做逻辑式编程，代码风格和命令式一样，只是在思考过程的时候，更偏重于逻辑思想部分，对于复杂的问题可能会更有优势。

命令式编程是计算机硬件的抽象，可以这样理解，我们把预先写好的一条条代码翻译成机器语言就是01指令，告诉计算机先执行哪个命令后执行哪个命令，最终是去改变了计算机硬件的稳定状态 1 稳定 0 不稳定 所以，可以说命令式编程是计算机硬件的抽象。

函数式编程是对于数学的抽象，理论基础来源于数学基础。

概念分析

闭包

定义：

> 一种特殊的函数，绑定了函数内部引用的所有变量，把它引用的东西都放在一个上下文中“包”了起来。百科定义：包含两层含义要执行的代码块（自由变量以及自由变量引用的对象）和自由变量的作用域。

闭包的概念在很多不同内容中都有提到，如数据库原理中函数依赖有闭包、JavaScript也有闭包、好多语言里面也会提到闭包的概念，它们得意思是否一样呢？答案是一样的，只是他们的表现形式不一样，数据库中的闭包是描述列是否可达、数据库范式的依据，把需要的东西放在了“上下文中”包了起来，解释闭包时先让我们来回顾一下前提条件：

函数式编程以函数作为一等公民，以函数为单元完成各种操作，函数是自变量到因变量的一一映射，即一个值到另一个值得映射，接收一个值返回另一个值，好了前提条件已经有了让我们来完成一个函数，计算两个数的相加操作，如下：


```
def plus(senior):
    return senior + 100

if  __name__ == '__main__':
    result = plus(3)
    print result
```

由于函数只能接收一个参数，该函数只能计算加100，这一种类型加法，想一想怎么才能让它再接收一个参数不变的情况下，计算出结果呢，也许你会想到先把值保存起来，不过函数式编程是不保存计算中间结果的，即变量是不可变的，我们可以把这个函数处理结果传递给另一个函数（同样接收一个参数），这样一来即可计算两个数加法，如下：

```
def plus(senior):
    def pluaA(second):
        return senior + second
    return pluaA

if  __name__ == '__main__':
    pluaA = plus(3)
    result = pluaA(2)
    print result
```

上面代码中plus即是一个闭包，闭包不仅仅是一个函数，从定义可以看出其包含几个部分，引用变量、代码块、作用域，可以结合上述代码加以理解。

闭包函数里面的函数虽然有函数名字但是并没有意义，只是一个名称而已从外面不能直接访问，属于匿名函数，个人理解它属于一种特殊的函数以及该函数包含的作用域里面包含的东西，闭包是一种看不见摸不着的东西，因此不好理解，通过闭包可以将n个函数相互连接起来，可以无限相互之间结果进程映射，如果没有闭包那么数学中的函数将会是一个个死的函数式子，闭包是函数式编程的灵活、是函数式编程的核心。

把代码绑定到闭包后，可以推迟到适当的时机再执行闭包，是一种对行为的建模手段，让我们把代码和上下文同时封装在单一结构，也就是闭包里面，以后执行。

高阶函数

高阶函数从字面上面理解除了普通函数功能还是高级内容，即可以把函数作为参数或者返回值的函数，从这个角度来说增强了函数处理能力，书上定义为可以接收函数为参数或者返回一个函数作为参数的函数。大家思考一下，高阶函数为什么可以这么设计，闭包在里面起到了什么作用？

匿名函数

匿名函数即lambda表达式函数，经常作为函数参数或者生成表达式的场景中，闭包中里面的函数可以看成是匿名函数来理解，上面计算两个数的加法还可以改为这样：


```
def plus2(senior):
    return lambda second:senior+second

if  __name__ == '__main__':
    pluaA2 = plus2(3)
    result = pluaA2(2)
    print result
```

这两种写法意思和作用是一样的，其实上面的PluaA也是匿名函数，只不过这样写会易于阅读和理解，匿名函数常用在比较简单的参数表达式中，使得代码简化、简洁，但是不要滥用匿名函数，如果代码中到处是匿名函数那么看起来会很不好理解。

柯里化

通俗的理解是将函数的参数变为一个参数的形式，函数式编程提倡柯里化编程，尽量编写一个参数的函数，优化方便，简化代码。

语法糖

指程序的可阅读性更高、可以给我们带来方便，更简洁的写法，提高开发编码效率
某种语言中添加了某种语法，这种语法对功能没有影响，但可以提高可阅读性、间接性等，则称这种语法为该语言的语法糖

性能优化

缓求值

是函数式编程语言的一种特性，这个特性也比较好理解，尽可能的推迟函数或表达式的计算过程，等到真正用到的时候才加载数据，类俗语hibernate框架中的懒加载，利用缓求值的方式来使用映射，可以产生更高效率的代码。

尾递归、尾调用

尾调用是在函数的尾部，调用另一个函数，因为函数是语言没有循环，递归很重要，对于递归需要来不断优化，一般采用尾调用或尾递归来优化。顾名思义，尾递归就是从最后开始计算,每递归一次就算出相应的结果,也就是说,函数调用出现在调用者函数的尾部，尾递归就是把当前的运算结果（或路径）放在参数里传给下层函数和普通递归区别在于内存占用。

总结

函数式编程其特性非常适合于大数据处理、科学计算、数据统计等业务，随着人工智能、机器学习、深度学习的流行正在变得重要起来，从面相对象思维到函数式思维的转变，是我们更好的面对人工智能领域问题的催化剂，它会使我们从细节中解救出来以数学的思维考虑问题，不管你是从事哪个领域工作，都有必要了解一下智能时代发展的方向，因为未来你从事的职业很可能被智能化取代。