**本文档核心感受，分布式可以借鉴————————**

**（spark ML也是在调用breeze，切分里面的函数，作分布式处理）**

**spark的ML中，各种model的fit/transform，其实很像我们能源管理项目的做法。**

**就是——调用breeze里的现成的算法——不过需要自定义数据处理，想着如何改进为分布式调用———再加上一些代码设计（很多接口，导致代码看起来非常复杂）**

**涉及分布式部分——可能不是完整的将breeze的类全部调用、串起来， 而是将其拆分，适用于分布式环境。**

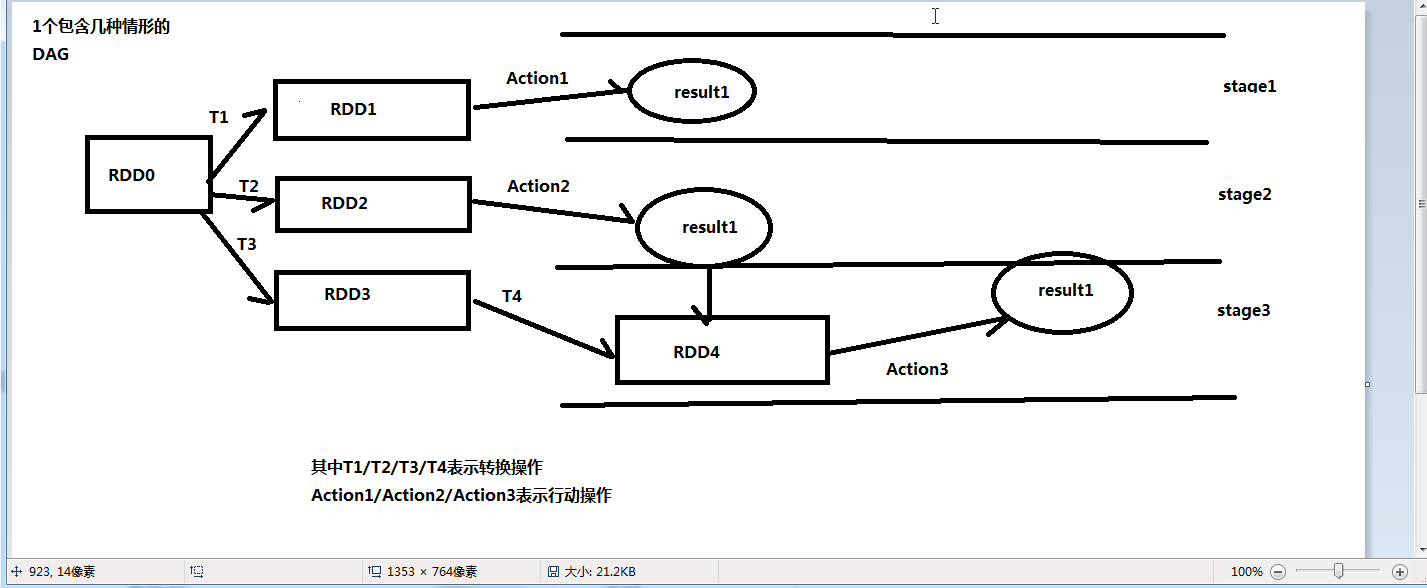
**为什么会如此复杂呢？接口、通用性**

**所以，分布式算法的关键在于——————真正弄清楚breeze库里面，单机算法怎么实现的，如何封装成分布式。**

**对于分布式RDD的DAG的重新理解：**

**另外：分布式不是“全程都是分布式、有时用分布式、有时不用”，不能简单这么认为，而是应该把程序分为：RDD(或DataFrame、DataSet)内、RDD（或。。。）外。**

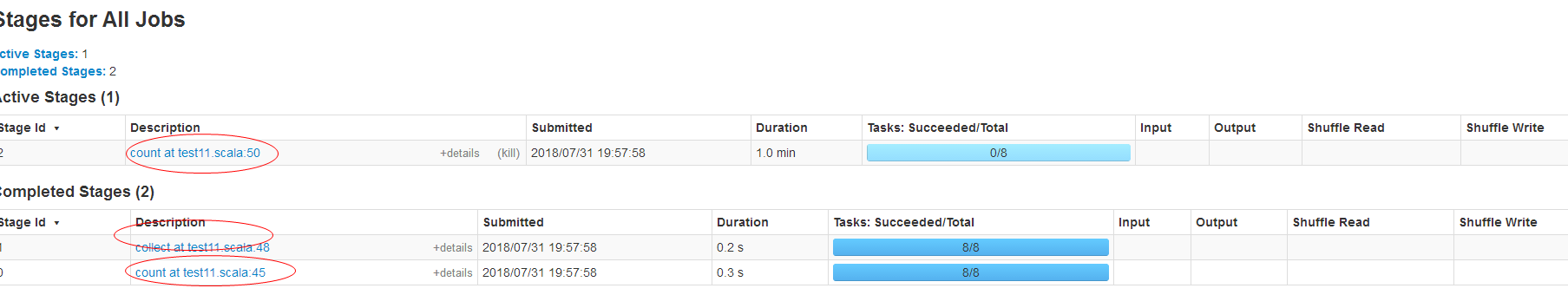
**RDD1,RDD2,RDD3……RDD(i)…… ，这些RDD，有的会被用到多次，而stage是根据action操作们进行划分， 一个action操作，回导致一次真正的运行与计算。 但action的结果，有的是最终结果，不在被其他RDD用到；有的则还是中间结果，会被其他RDD用到——这个不知道是怎么生成DAG的，如下所示：**



亲测程序：

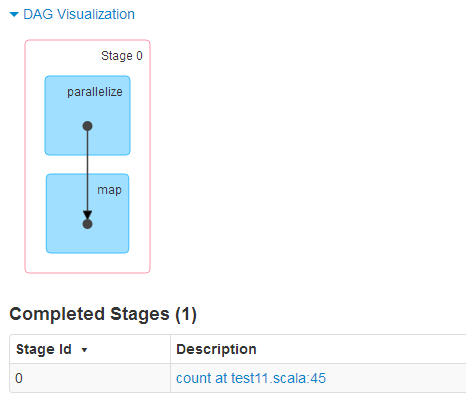


stage图：

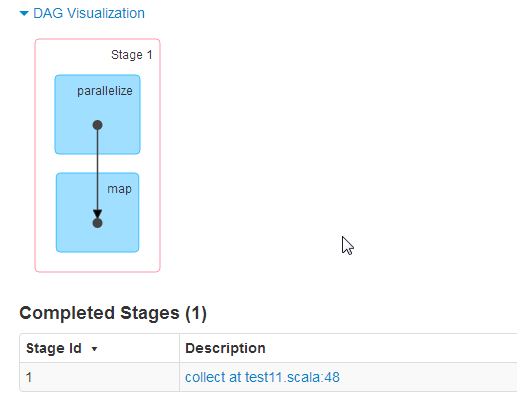


DAG图示：

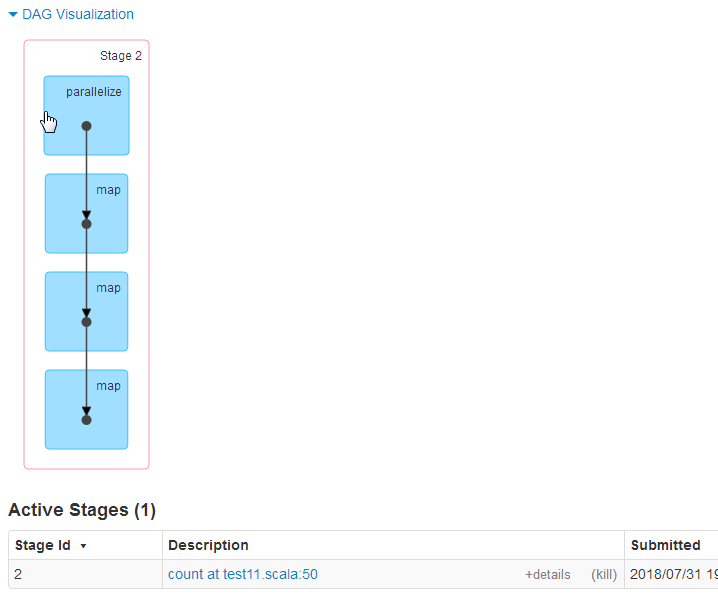
stage0的DAG



stage1的DAG：



stage2的图示：



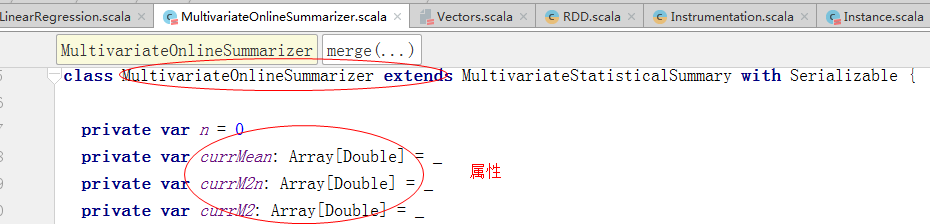
spark的一个编程技巧：

一个class，定义一堆var， var里面是各种属性值

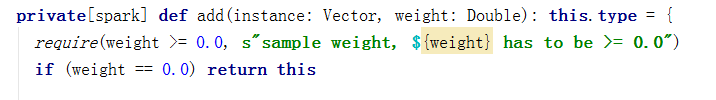
class里定义各种方法func， 但是func返回this.type ，这样func做的只是，不断更新var的值，而class的实例可以使用函数式编程（不断的调用obj1.func1.func2.func3….）

也即，方法只是为了属性值的更新，而定义的操作。实例如下，spark对一条样本（VectorX,y）进行统计量计算：

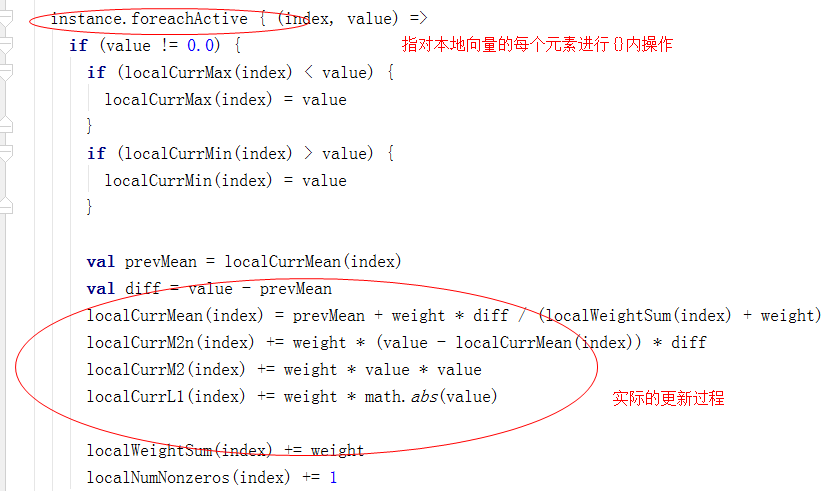
先定义属性——



再定义方法，不断更新属性值（计算过程），比如add方法：



及



LinearRegression的逻辑：

spark-2.2.0\mllib\src\main\scala\org\apache\spark\ml\regression\LinearRegression.scala

1. 其train方法：

  
关键逻辑：

计算描述统计量部分：

使用treeAggregate，(分区内用seqOp,分区为用combOp)具体作用见：

C:\Users\Administrator\Desktop\学习笔记\spark\RDD\aggregate和treeAggregate区别.txt

其中Instance是一个case class类，代表1个样本实例（1条记录x11,x21,x31,x41 … y1）

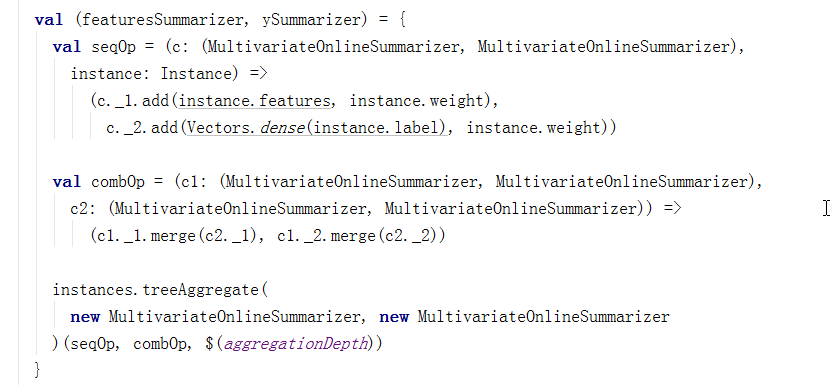
其中初始传入的值是一个tuple2，每个元素为“描述统计类”，

seqOp为对使用MultivariateOnlineSummarizer类，进行add操作，更新该对象的属性，注： seqOp是 (U,T)=>U, T是RDD里的元素，U是seqOp初始接收的类型

combOp是对每个元组进行merge操作。

最后得到tuple2，每个元素为“描述统计类”。

但此时的描述统计类的属性值，已经全部经过add/merge操作得到更新。



接下来：定义损失函数、优化算子————

注意：只是定义，即构造这两个类（只有初始属性，还没有调用方法来计算、更新属性），这是spark和scala的编程习惯。

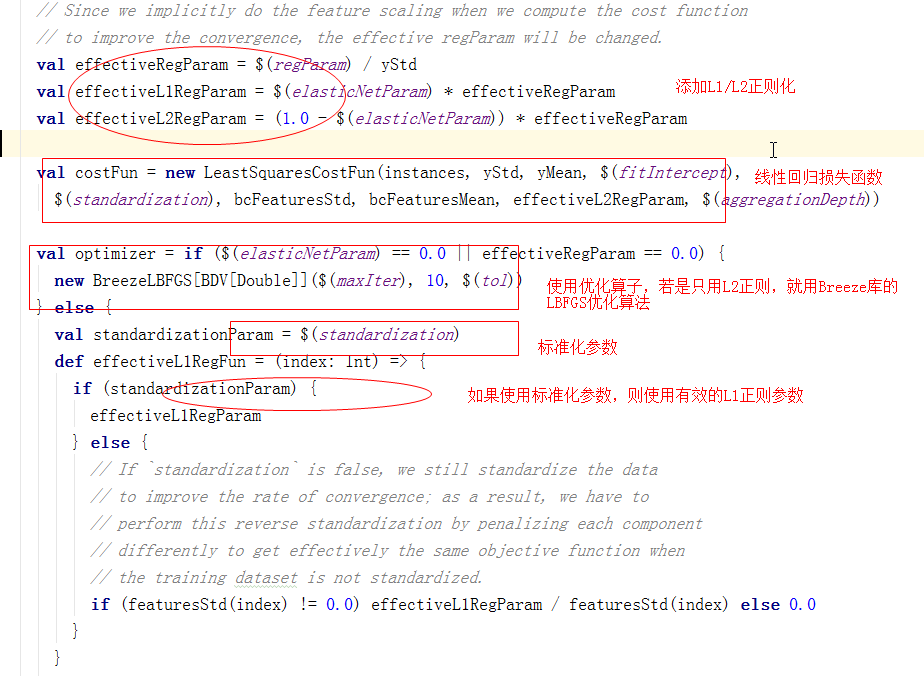
elasticNetParam:L2和L1的惩罚项比例系数，为0则只加入L2，为1则只加入L1，否则为混合： 即 正则项L = α \* L2 + （1 - α）\*L1

regParam:应该是加在L前面的参数lambda

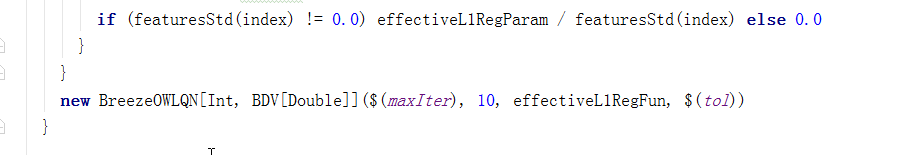
**优化算法选择**：如下——————————————

1.当elasticNetParam为0或者regParam=0时（即只用L2或根本不用L时，也即不用L1时），使用Breeze的LBFGS算法。

2.否则，使用Breeze的OWLQN算法



接上：

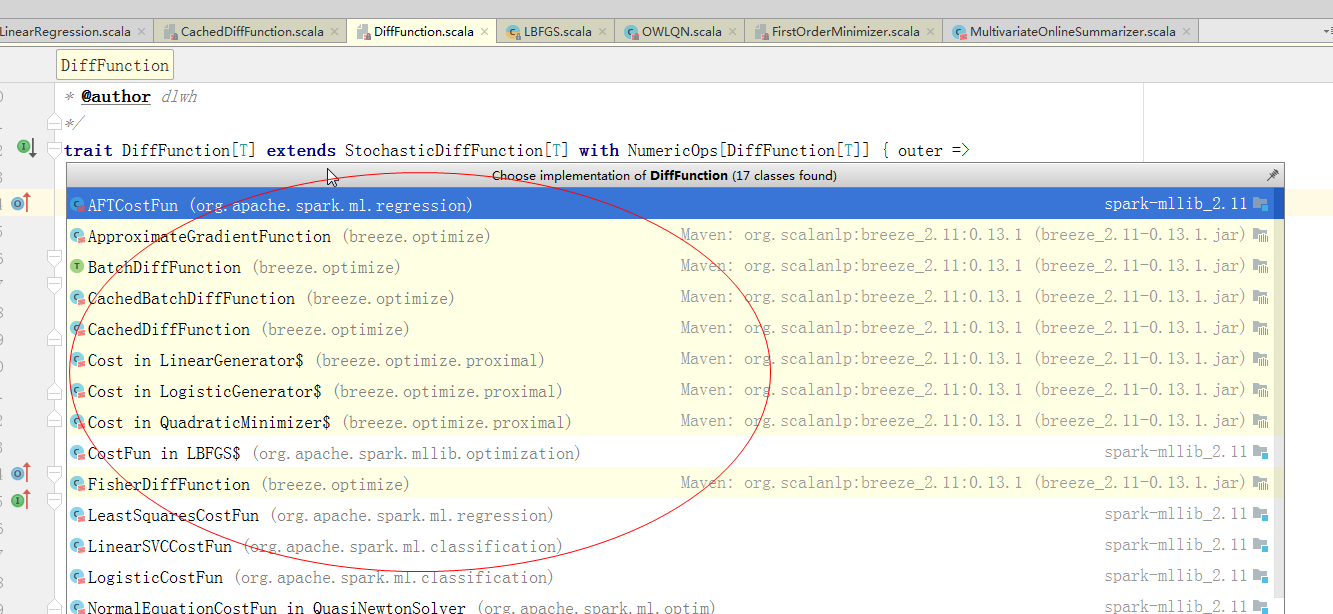


另外，损失函数如下：

是一个Trait——DiffFunction

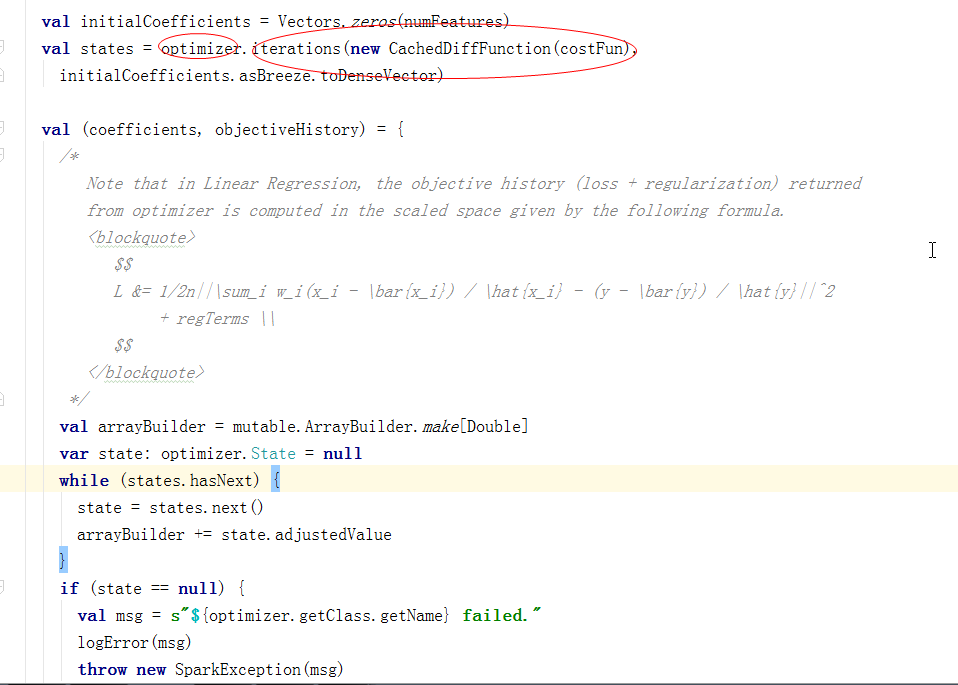
具体是调用的breeze的optimize包





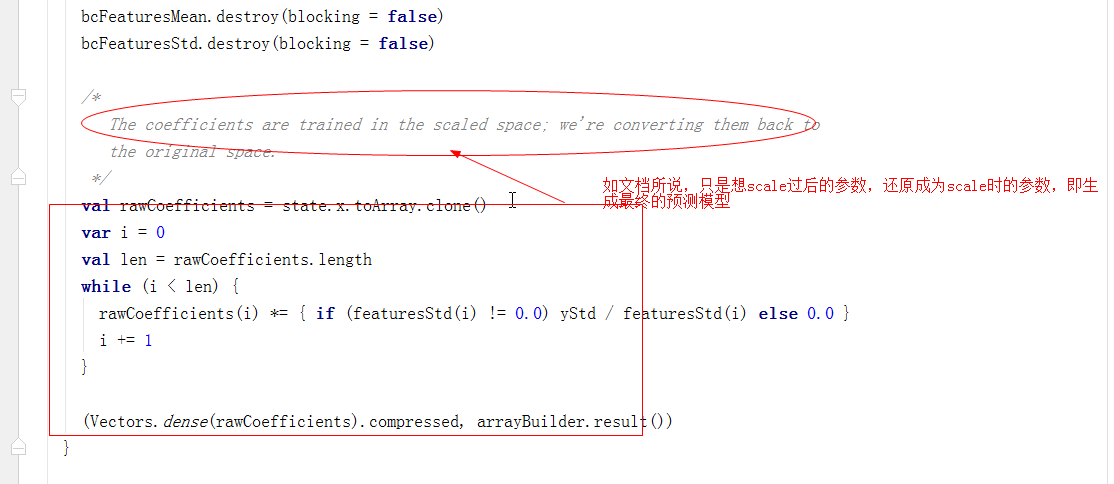
真正的更新权重（优化损失函数部分）如下：

在于optimizer.iterations方法



下面的部分，不是很核心：

1.参数还原（上面计算的系数，是标准化空间计算得到的参数，不是最终用于预测的原始模型的参数，需要还原成原来的特征空间的系数）



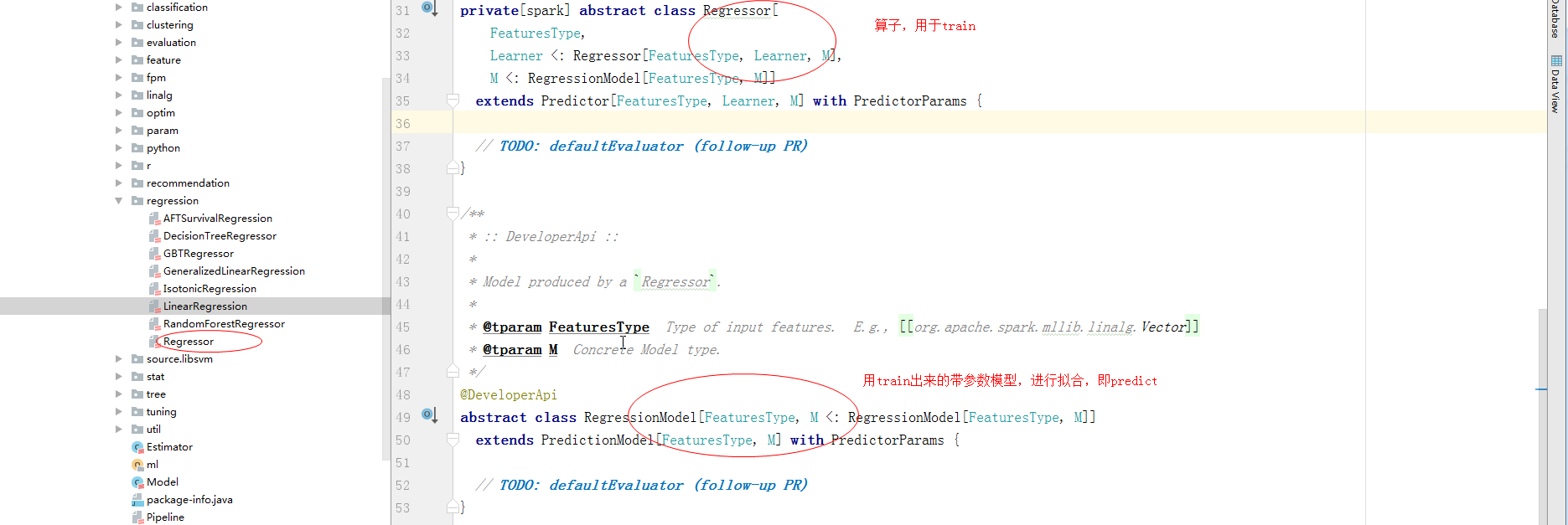
以及剩余部分：



**关于回归类的算子，均继承自两个抽象类Regressor、RegressionModel类**

**Regressor：用于模型训练，其中调用了继承自其父类但由其子类覆写override的fit方法，fit方法封装了回归的不同实现类的train方法**

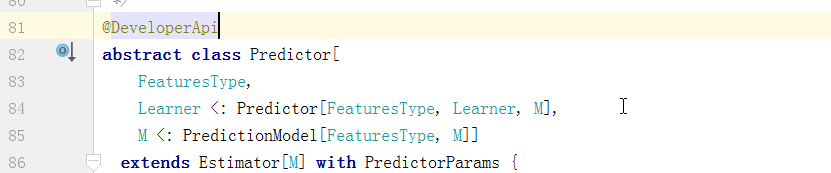
**RegressionModel:用于模型预测，也像Regressor一样调用了继承自其父类，但由其子类覆写override的transform方法，transform方法则封装了predict方法，predict是真正的预测逻辑**



**由于fit是更高一级的概念，不一定非得是训练+预测，还有那种算子类**

**fit阶段：**

另外,LinearRegression类只能用fit，其fit是继承自Regressor类，并继承后者父类（抽象类）Predictor的fit方法，没有override：



该抽象类有fit方法：非抽象方法：

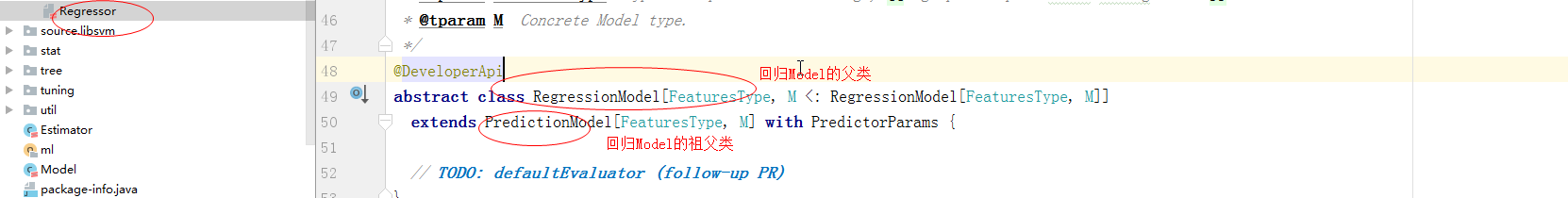
所有的SomeModel(比如LinearRegressionModel, Some表示Model前缀),在fit阶段，都是执行了其自身的train方法的实现——而Some类的fit方法，都是继承自其祖父类Predictor的fit方法（没有覆盖，直接继承获得），fit里有实际调用了Predictor的train方法——该train方法，则是各种Some类（Predictor的实现类）的具体不同的train方法——即模型的真正训练部分



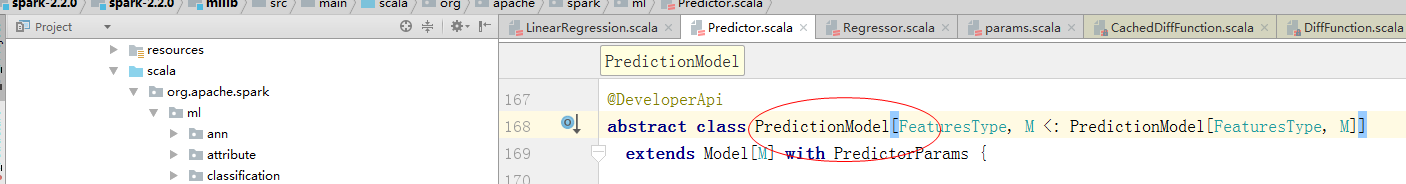
其中的train方法，真正实现是在LinearRegression里的train方法，即我上面分析的部分。



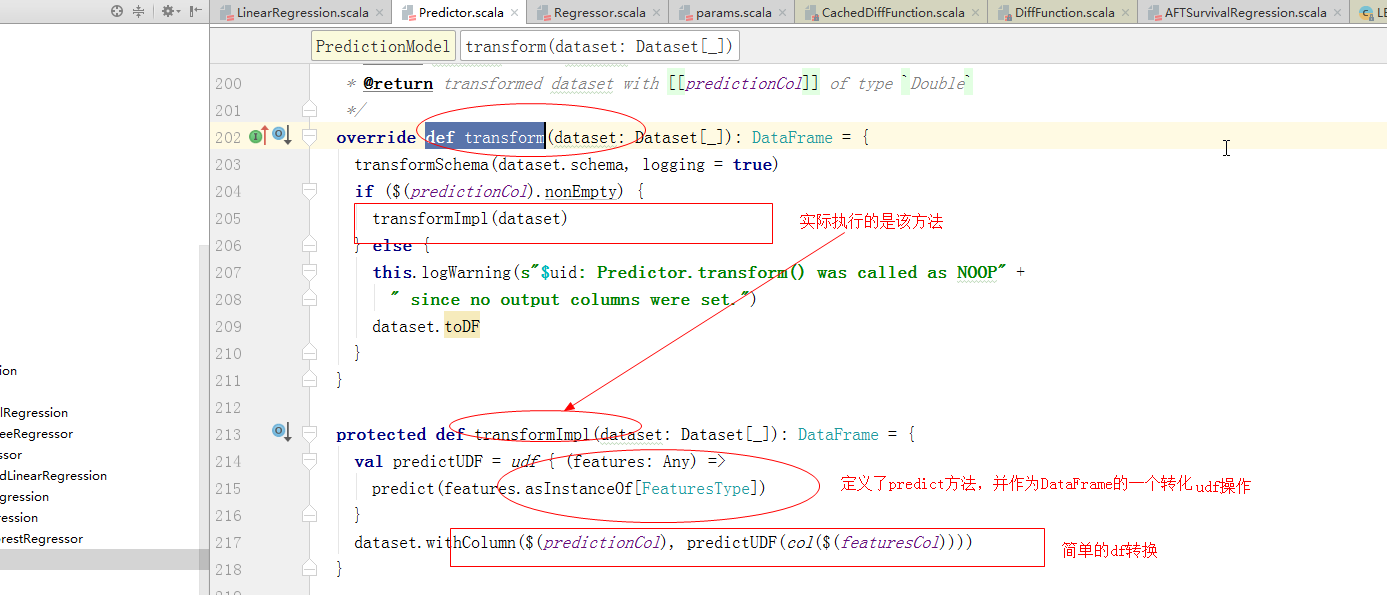
而transform阶段：



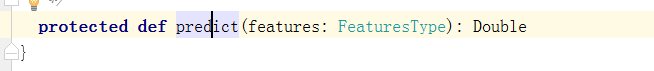
方法继承自：



的

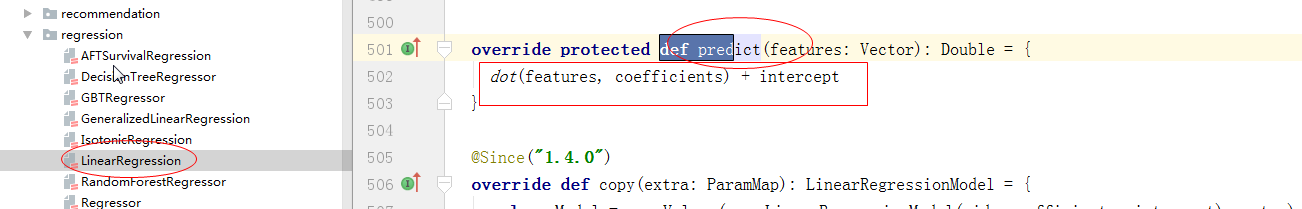


该trait的predict方法接口如下：



线性回归的实际算子则为————简单的矩阵与向量之间的点乘——y = X\*β：

如下：



dot使用的是BLAS库的点乘dot方法：