# SparkTimeSeries源码解读

## TimeSeriesRDD原理解读

1. TimeSeriesRDD继承自RDD，存储了一个基本的RDD和时间的index信息
2. TimeSeriesRDD的创建

一个parent: RDD[(K, Vector)]

一个index，该index起始并不在数据节点保存，也就是没有在RDD中作为数据，只是TimeSeriesRDD将两者封装起来了。

1. TimeSeriesRDD里面继承了RDD的一般方法，同时实现了自己的一些特有方法，findSeries、differences、quotients、slice、fill、mapSeries、toInstants、seriesStats、collectAsTimeSeries

findSeries根据key查找序列

differences差分

quotients作商

slice从整个RDD中选取时间序列的子序列，根据RDD选取

fill缺失值选取

mapSeries实现了TimeSeriesRDD内部元素的map

toInstants以及类似的一系列方法，将TimeSeriesRDD转化为一般的RDD或DataFrame方便和其他对接，其中也设计到了分区方法。

seriesStats实现mllib中的基本信息统计

上述方法中有些是对单变量时间序列的，有些是对所有时间序列的。

像lag、findSeries就是针对所有时间序列的

而fill等就是针对单变量时间序列的。

1. TimeSeriesRDD的原理

对每个变量进行列式存储，如果多个变量，可以实现并行化。但如果同一个key的多个变量间进行互动则需要很多的shuffle。因此应该拆分开，如果只是需要简单的时间序列操作，如lag，求自相关等可以基于该设计，而进行大量的多变量互动可以基于行式存储。

1. 统一以下时间序列的标识，方便后面理解

如果（\*）标识一个时间序列，其中表示时间维度，可以表示不同的变量或者不同的个体。

1. TimeSeriesRDD的设计

一个全局的TimeIndex信息

一个RDD[(K, Vector)]，其中Vector为对应的序列，K为对应的序列的key。

对于基于RDD的时间序列一个很常见的**误解**可能就是它的时间基于RDD[time, Vector [value]]的形式，也就是RDD的行轴依据的是时间，也就是\*式中的，列轴依据的是变量id或者个体id，也就是\*式中的。

**但，我们发现SparkTimeSeries并没有在时间维度上并行化，而是在变量或者个体维度上实现了并行化。也就是RDD的行轴依据的是变量id或个体id，也就是\*式中的，而列轴依据的是时间，也就是\*式中的时间序列函数剖析**。

1. 根据2）我们发现TimeSeriesRDD的列式存储的核心其实就是把数据的行和列的逻辑进行了调换，其基本的实现是依据RDD的行式存储，把行和列的数据概念换了，RDD进而就编程列式存储了。
2. TimeSeriesRDD的内置函数的实现原理

TimeSeriesRDD就是分布式的对其中每个用Vector存储的序列调用函数的过程。

1. 一般map类型的

如lag、differences 、slice、Arima时间序列模型等等。

TimeSeriesRDD会分两步走：

1. 一步是基本RDD[(K, Vector)]的变换，将RDD[(K, Vector)]中的(K, Vector)转换为TimeSeries[K]，然后进行lag处理（单机版的），然后再将lag的TimeSeries[K]转为SeqLike[Vector]的形式，最后通过RDD将其抹平
2. 另一步将TimeIndex进行对应的处理
3. 一般filter类型的

如findSeries

1. 设计到shuffle的
2. TimeSeriesRDD

设计到多变量的大量迭代的框架就不能只用TimeSeriesRDD，而应该在同时引入一般RDD进行编程。

如一个形式为time+key~variables的数据，其中variables为变量，形如[variable1, variable2, …]我们需要处理流程是时间规整 => lag&&平滑在=> mllib的机器学习训练和预测，那么一个比较好的策略是这样：

1. 先将数据变为列行轴为key+variables~time的形式
2. 根据需要的规则，制作TimeIndex，结合1）生成TimeSeriesRDD，实现数据在时间维度上的规整
3. Lag&&平滑
4. 将数据转为

进行lag和平滑之后，再将TimeSeriesRDD改为