**spark中图要素：**

顶点：VertexId:Long

边：Edge(src:VertexID,target:VertexID)

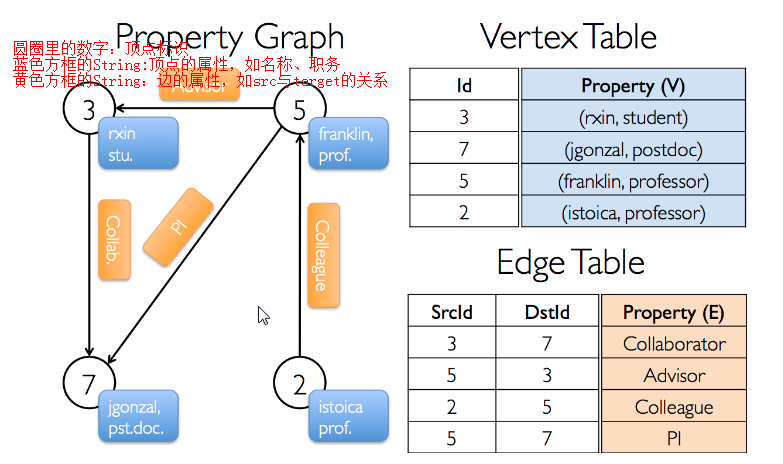
顶点和边都可以添加属性

边还可以添加权重。

**Property Graph**

属性图： Graph = (V,D),其中顶点集V的元素顶点，带有属性（其标识必须是Long型），而绑定了其它属性给顶点、给边也绑定了属性，就是个属性图

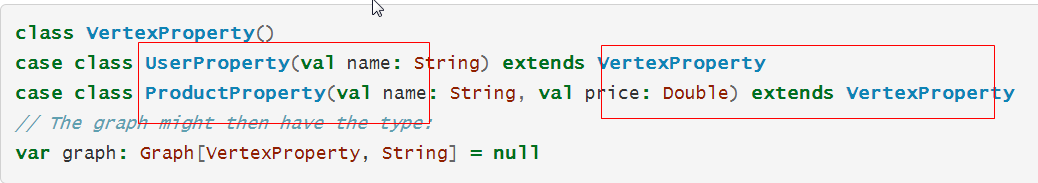
如下：



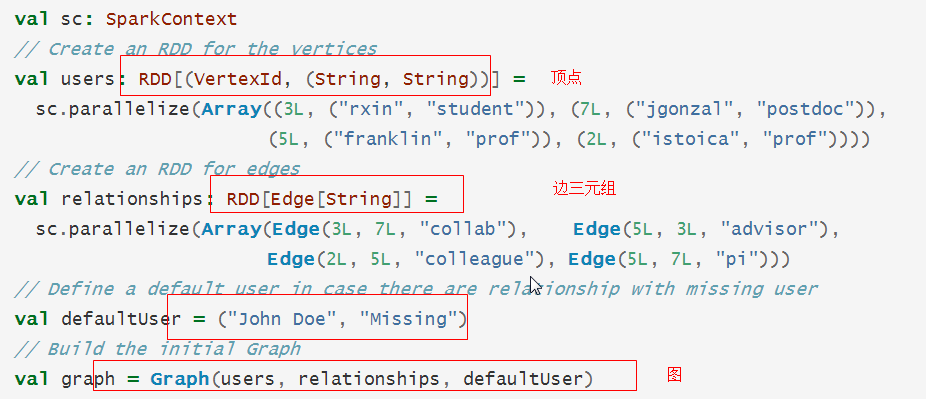
属性不是必须的：

但顶点标识、SrcId、DstId是一个图最基本的，必须的。即右侧图的白色区域字段。

顶点属性（用户自定义属性）可以通过继承实现：



由RDD构造图的一个示例：

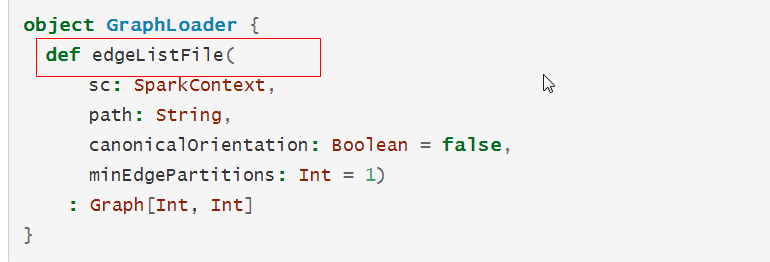


Graph构造的方法：

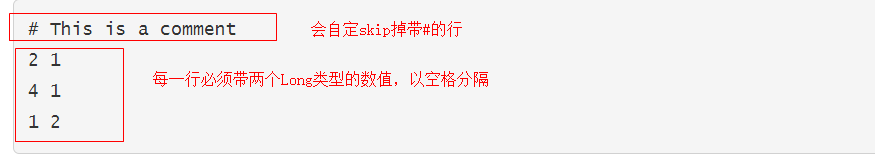
可以由边构造（则边中包含的顶点，默认为Graph中的顶点）

总体是以下2种方法：

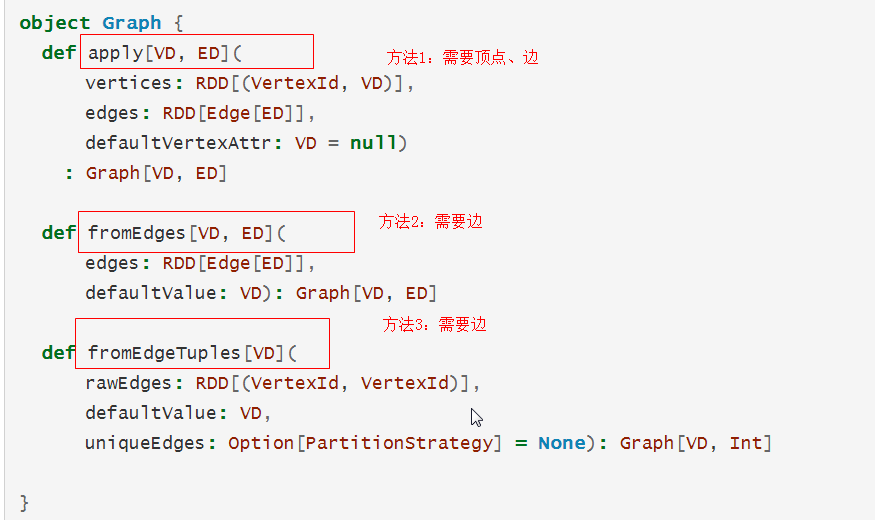
1. 从文件构造（支持hdfs，构造出的图的edges分区默认不变，和hdfs的block分区相同）——GraphLoader.edgeListFile方法



文件形如：



2．Graph类构造：3种方法

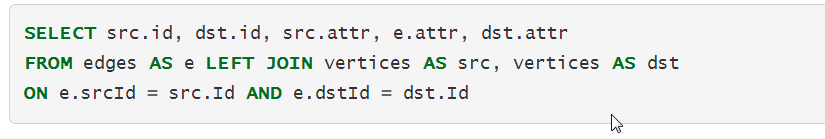


图还有3元组视图,也即图可以表示为一个3元组的RDD：



图其实是3元组里的所有信息连接起来的，连接逻辑如下：

将src、dst的边、属性，以及edge的属性连起来。



# Graph Operators

graph的算子：在Graph、GraphOps两个类里：

具体见spark API:

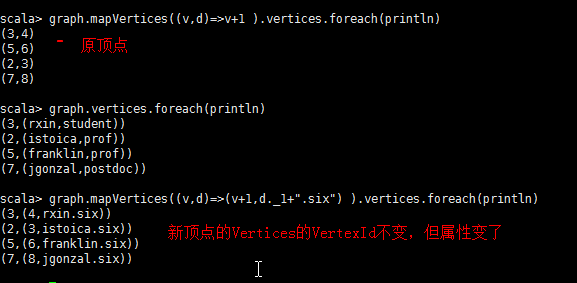
http://spark.apache.org/docs/2.2.0/api/scala/index.html#org.apache.spark.graphx.Graph

<http://spark.apache.org/docs/2.2.0/api/scala/index.html#org.apache.spark.graphx.GraphOps>

mapVertices方法，顶点的Id不变，属性会变：



这里的VD即是属性，不管是啥形式的。只会有VD生成VD2，VertexId不变。



同样mapEdges也是只会改变Edge的属性，不会改变其所在分区号、srcId、dstId

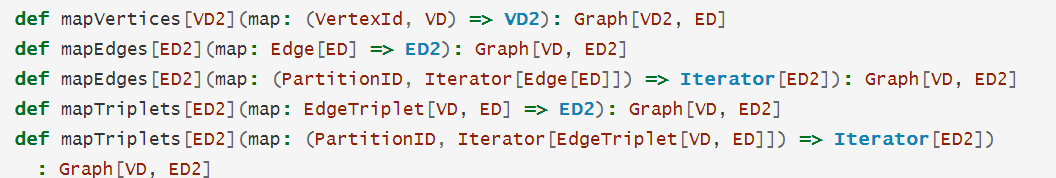


操作如下：

mapVertices、mapEdges仅修改属性、不修改标识，不改变图的结构



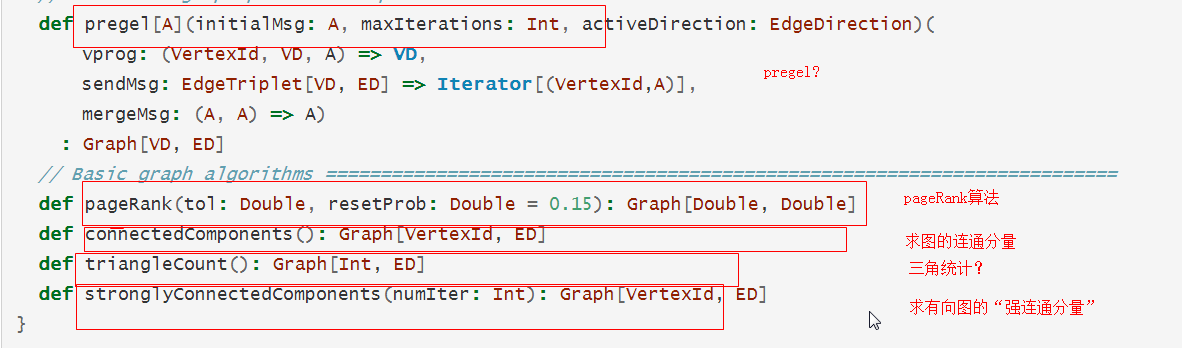
顶点、边、边三元组的map操作



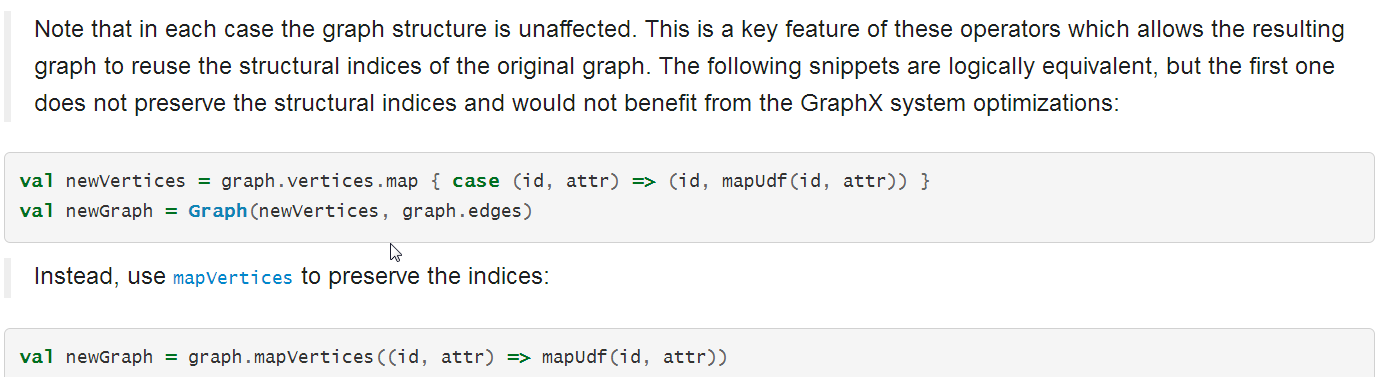
下面的几个比较复杂：



几个比较有名的算法——也是Graph类自带的：

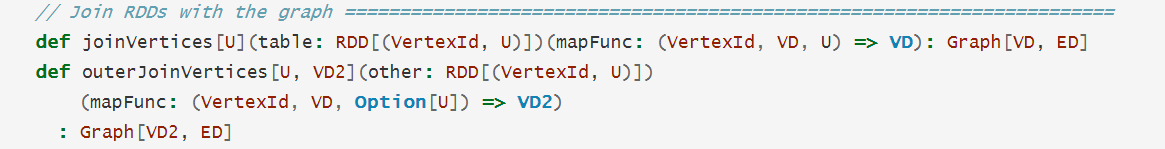


使用Graph.mapVertices相对于RDD[Vertex].map的好处，可以复用原图，因为前者不改变图结构，只改变属性；而后者必须生成新的图：

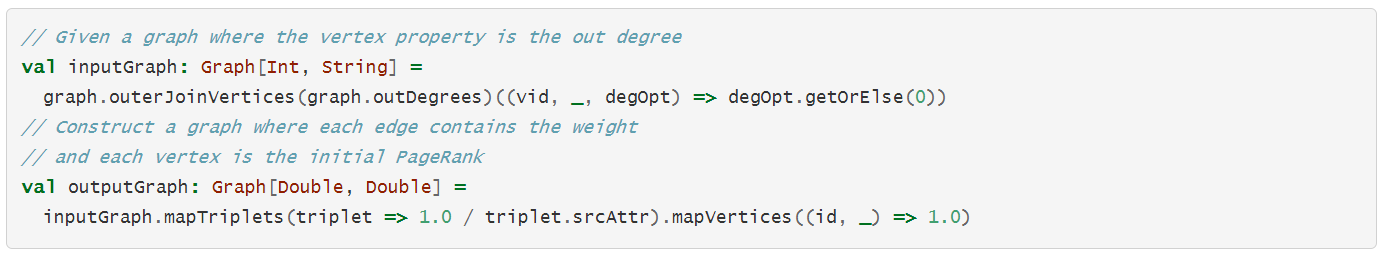


join操作的作用： 类似于sql，连接RDD和Graph时，以他们的VertexID为键，将RDD中的U(是一种类型)，作为Graph的新添加的属性

简言之：joinVertices就是给Graph的顶点，添加新的属性



一个示例：



关于Graph的构造：

当RDD[Edge[ED]]中，其srcId和dstId的集合中，若在RDD[(VertexId,VD)]里的顶点中没有前者（边的顶点集合）时，该顶点应该有个“默认属性”， 需要用一个元组定义，按顺序自定义值——————顶点有N个属性。则自定义一个N元组。

为出现在 RDD[(VertexId,VD)]里的顶点，其属性都用自定义元组里的属性，赋值。

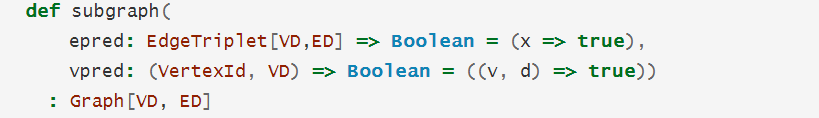


triplet的属性：

一个Triplet(三元组),有以下属性： srcAttr(src顶点的属性，是元组)、dstAttr（dst顶点的属性，是元组）、attr（边的属性，是元组）



subGraph使用——生成子图：



传入“过滤函数”（返回bool值的那种）——可以对顶点、边三元组（EdgeTriplet）的至少一种进行过滤。

即至少要传一个“过滤函数”. 如果不传对应的过滤函数，则默认为true。 如果啥都不传，就是原Graph

如下所示：

