**graphx之图迭代**

原创 2016年09月12日 19:57:37

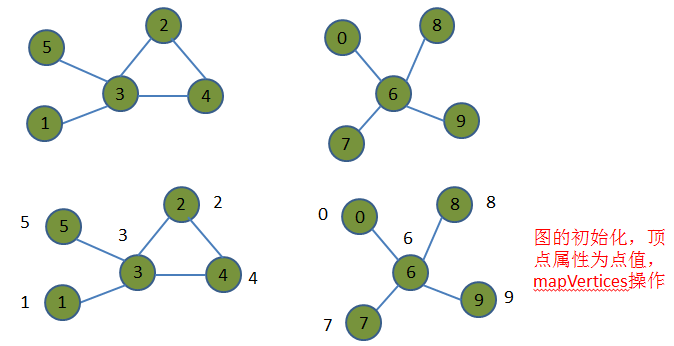
* 标签：
* [spark](http://so.csdn.net/so/search/s.do?q=spark&t=blog) /
* [graphx](http://so.csdn.net/so/search/s.do?q=graphx&t=blog) /
* [连通分量](http://so.csdn.net/so/search/s.do?q=%E8%BF%9E%E9%80%9A%E5%88%86%E9%87%8F&t=blog) /
* [迭代](http://so.csdn.net/so/search/s.do?q=%E8%BF%AD%E4%BB%A3&t=blog)
* 1057

**迭代**

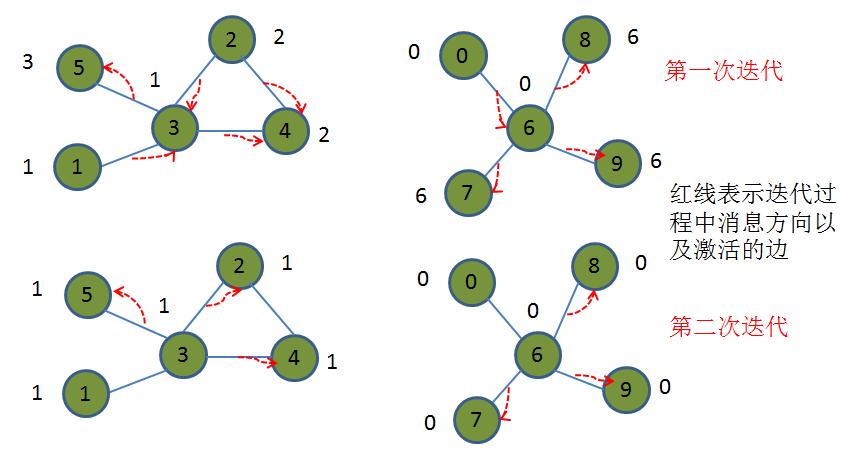
迭代思想是spark的精髓之一，所谓迭代，即每一步的输出结果作为下一步的输入，因而相邻迭代具有很强的关系。   
graphx更是将这种迭代思想运用的灵活高效。

**联通分量**

通过graphx lib中的ConnectedComponents连通分量算法，简单介绍迭代和消息传播机制。示例见下图：



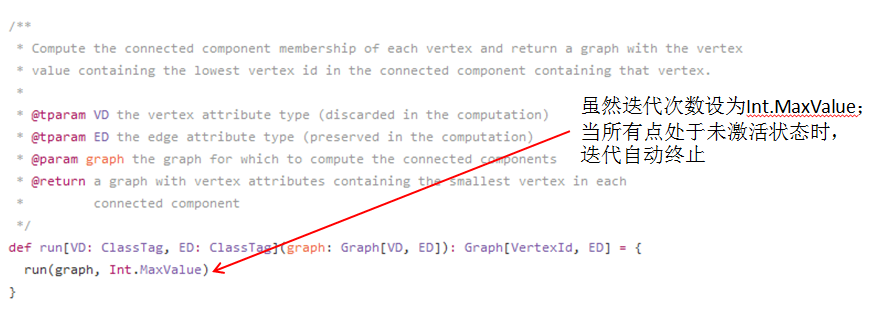
在完成顶点的初始化后，连通分支开始迭代过程：   
为区分顶点自身id与连通分支id，后者称作cid。   
1. 发送消息：若点的cid小于邻居顶点cid，则向邻居顶点发送自己的cid   
2. 接收消息：若邻居顶点cid小于点cid，会接收到邻居顶点发送给自己的cid   
3. 消息合并：选择所有消息中最小cid作为自身新的cid。



上图中，经过两轮迭代，已得到各连通分支，具有相同cid的点属于同一连通分支。

**ConnectedComponents剖析**

关于联通分量的介绍，以spark2.0中连通分量算法的源码为例：

也许介绍到这里，你多少已理解消息传递过程，以及连通分量是如何实现的；可能对pregel模型并不是很懂；没事，下篇博客会认真梳理pregel迭代模型，希望对你有所帮助