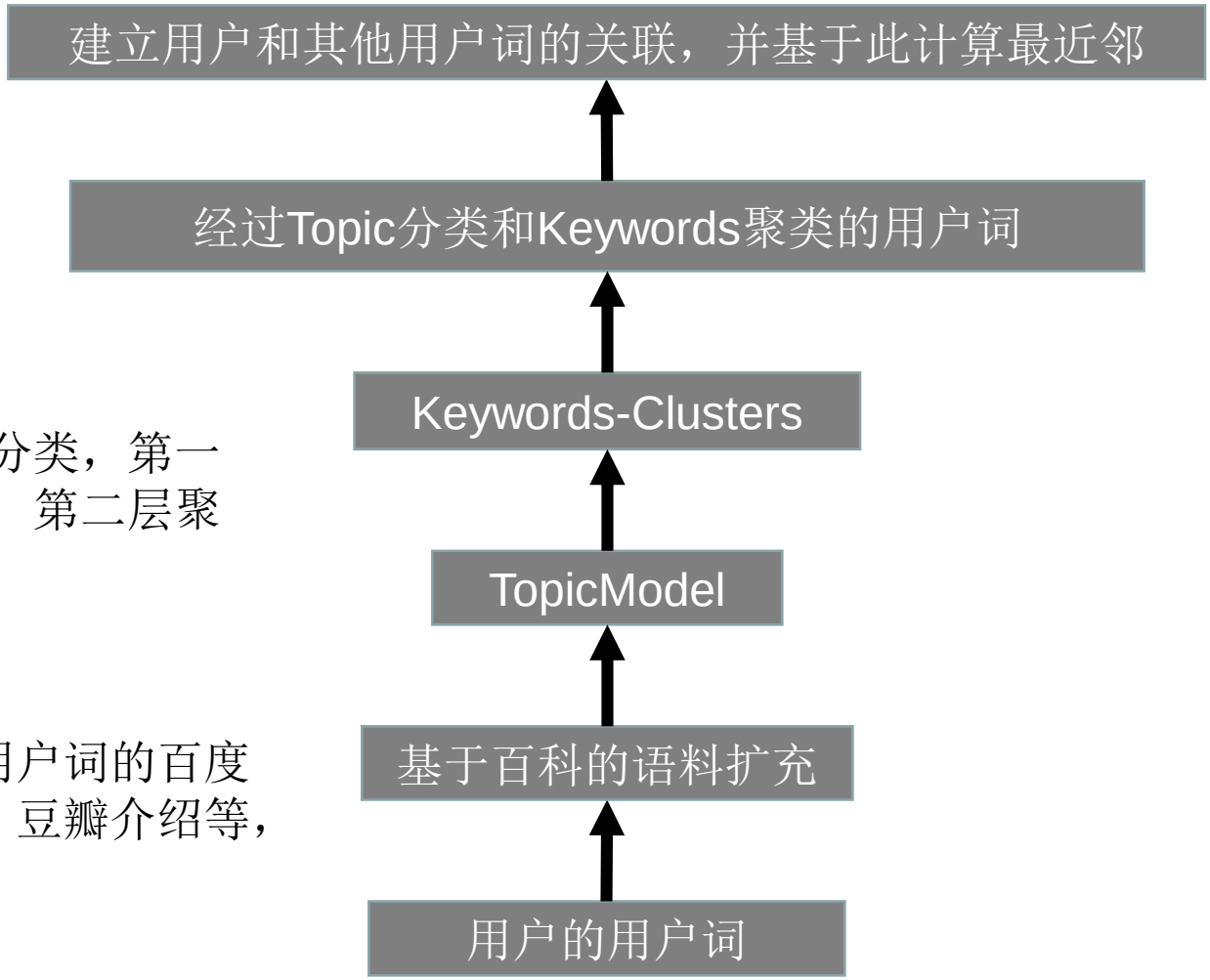
ToYou的相似度计算模型

以下是简化的层次视图

本来用户只和10个词有关，通过词聚类，用户现在和很多词有关了；



两个层次的分类，第一层主题分类，第二层聚类

搜索关于用户词的百度百科页面，豆瓣介绍等，作为语料

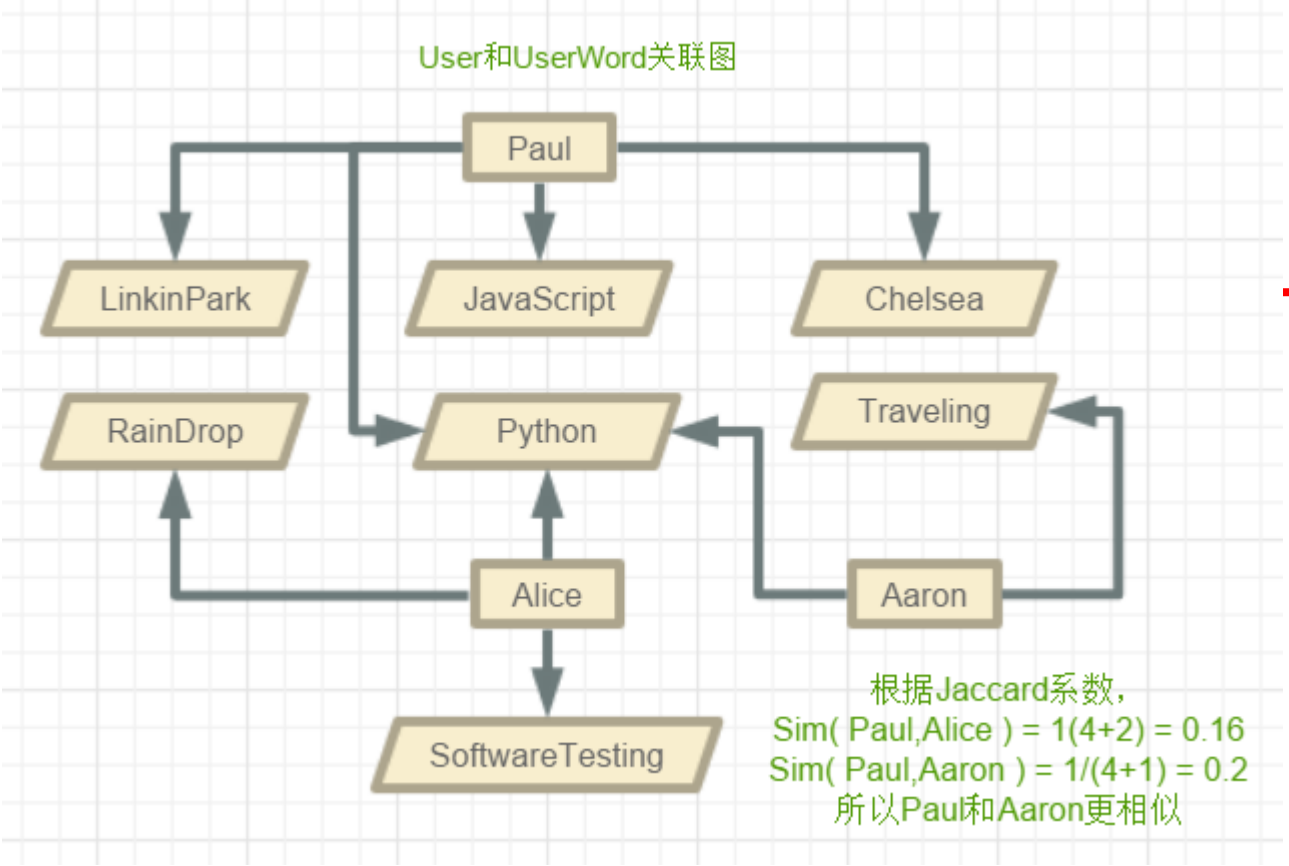
提供Item相似度的KCC模型

用户使用若干个词语（不超过10个）来表征自己；

那么如何利用这些词语，来计算用户之间的相似度？

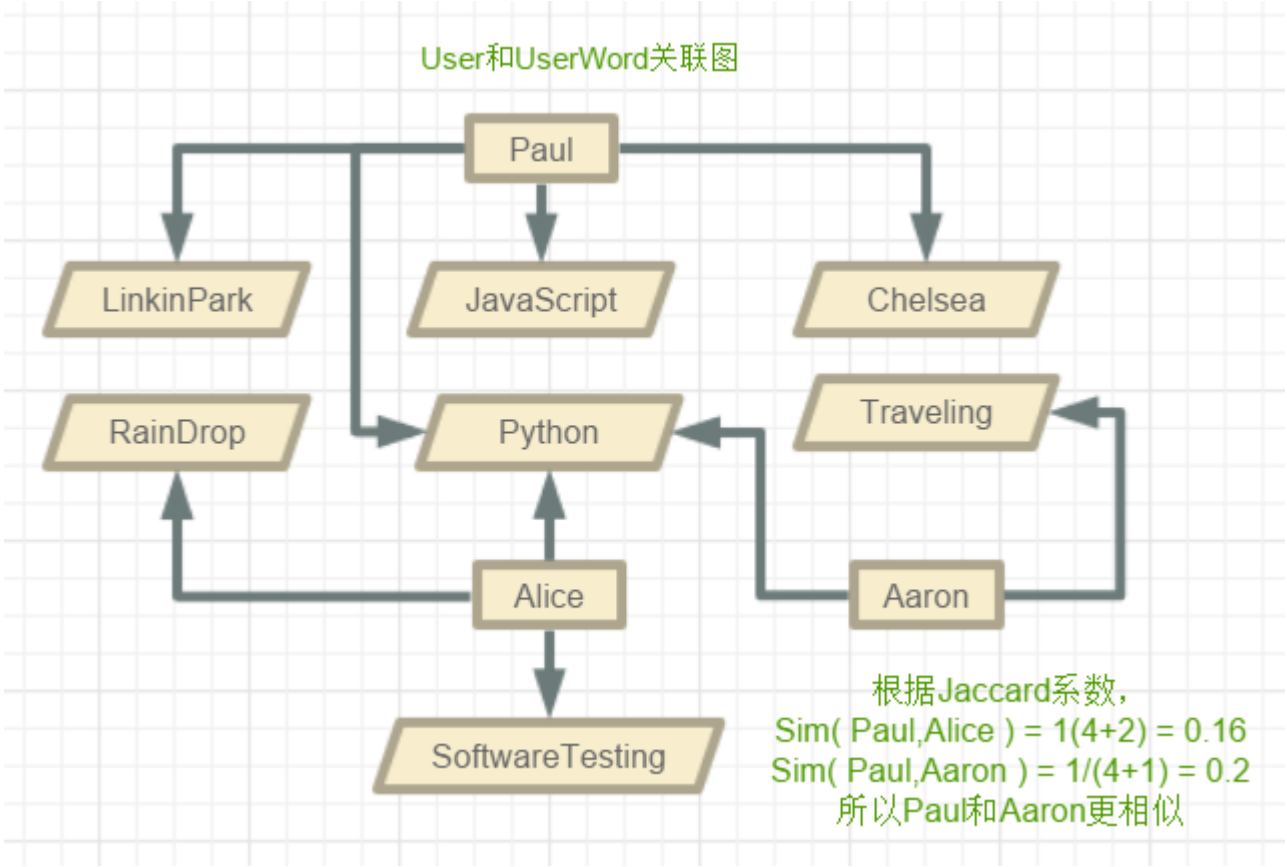
我们会使用图数据库，对User和UserWord之间的关联进行存储，存储模型如下图所示；
然后通过Jaccard系数，计算和某个User相关度最高的User；
SimVal = (UserA.Words ∩ UserB.Words) / (UserA.Words ∪ UserB.Words)

如下图所示，直接从显示关联进行计算的结果是Aaron相似度更高



但是，仅仅通过显示关联的词是不够的，正如上图所示的，用户之间显示连接的词可能会非常稀疏，因此需要挖掘UserWord之间的语义关联；

注意到这里的用户Alice的RainDrop是一首音乐的名字，同时我们发现Alice还和SoftwareTesting有关联，也就是说Alice喜欢音乐，而且从事软件开发和测试工作，这两点和Paul是非常相关的，
相反Aaron的另一个词Traveling和Paul就没什么关联，因此我们认为，事实上Alice和Paul的关联度会大于Aaron；

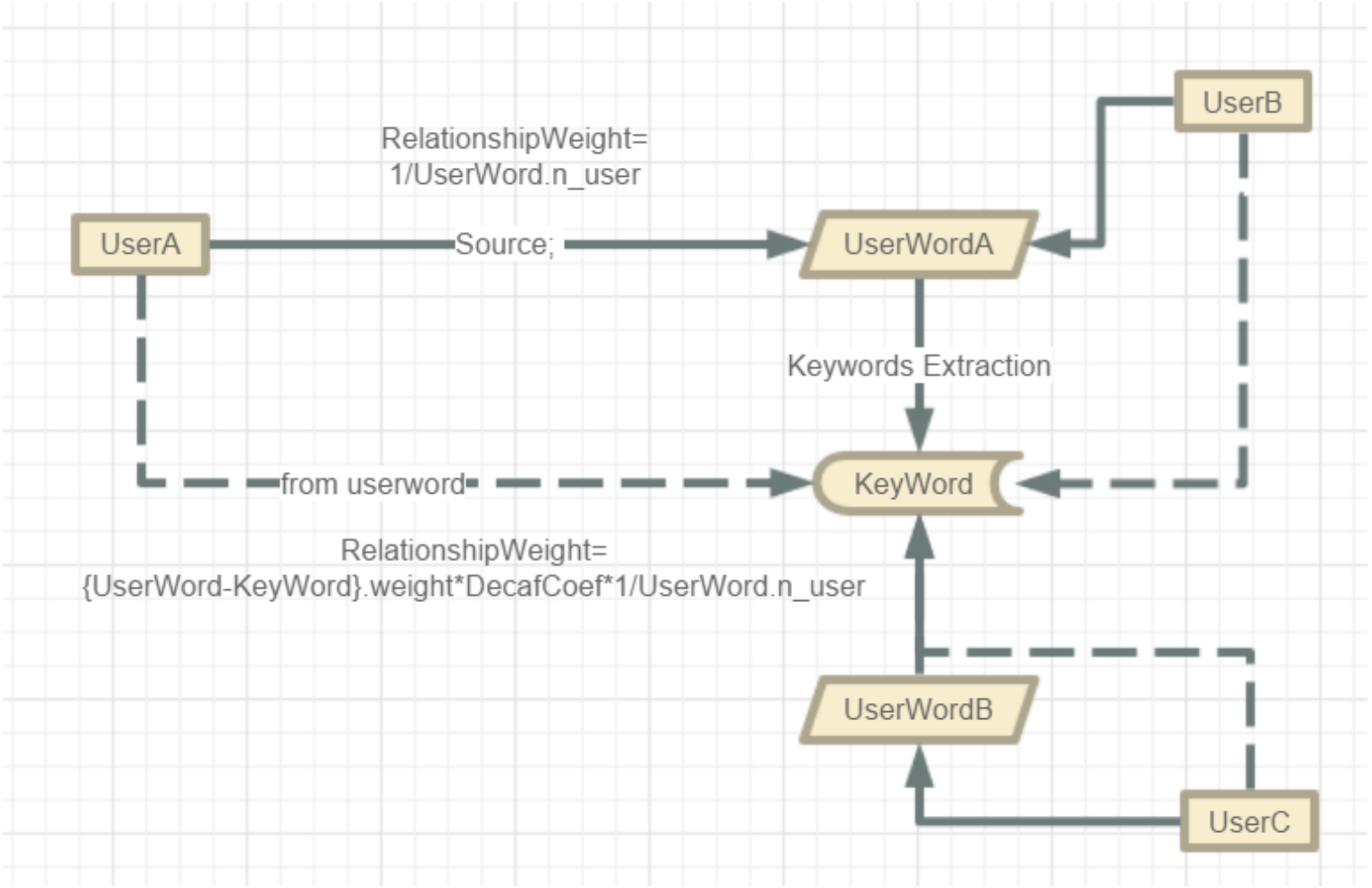


那么问题就在于，如何利用两个用户的非共有词，计算相似度？

我们将其称为，非共有词相似度；

受WordNet启发，我们认为应该首先计算词语之间的关联，再基于词语之间的关联，计算用户的相似度；

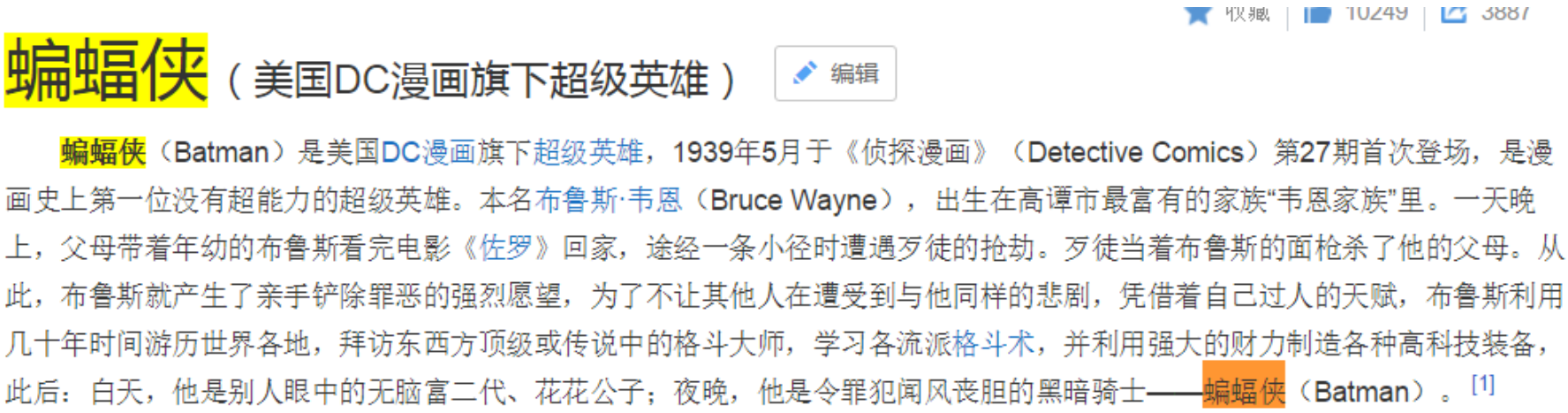
一个最简单的方法如下图所示，首先挖掘和UserWord有很强关联的Keyword，然后建立User和这些Keyword的关联，再通过这些Keywords建立User之间的关联；



一种提取UserWord相关的关键词的方法是，将Wikipedia或者百度百科的相关页面作为document，并使用常规的关键词提取算法对document提取关键词；

考虑到会有一些相关度不是很高的内容出现在页面中，因此可以duplicate关键词所出现的句子，以提高关键词所出现句子的权重；

关键词的数量可作为参数，另外提取的关键词权重需要被归一化；



上述非共有词相似度模型的本质是，使用用户和关键词的关联度，来表征用户，进而计算相似度；

然而我们认为这个方法的计算复杂性太高，而且不利于分布式计算和存储；
每个用户可能会和几十甚至几百的关键词产生关联，而这些关键词又会和非常多的用户产生关联，那么为一个用户计算最近邻的开销会变得很大；

因此我们提出两个改进：
首先应该先对UserWord进行分类，将UserWord划分到某个类，
比如属于动漫的词，属于音乐的词，属于人格描述的词等；

另一方面，我们认为应该对划分到一个类的UserWord进行聚类，这样可以大大减少计算量；
聚类基于UserWord的相关关键词；
聚类后，使用UserWord所在簇中的UserWordNeighbors进行User关联；

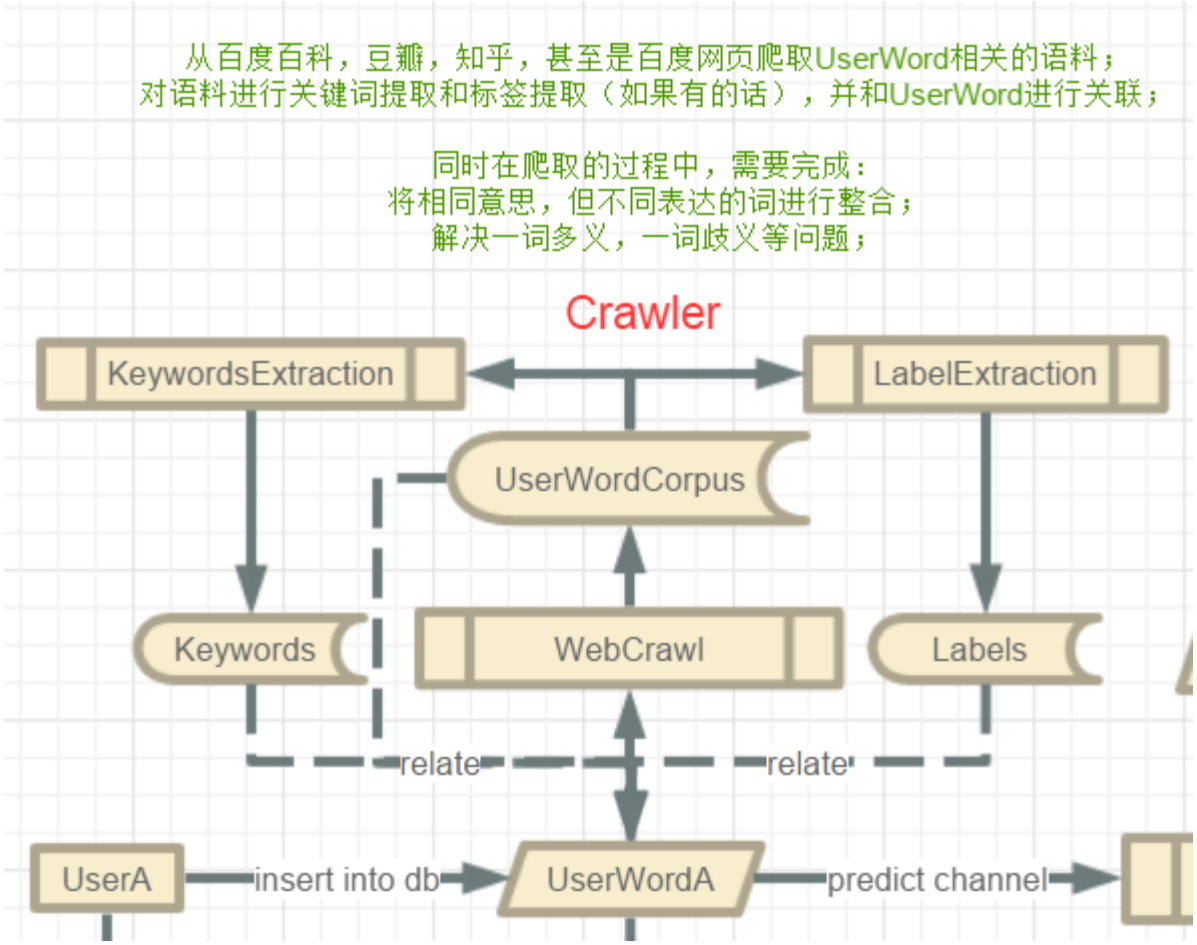
改进后的模型的本质是，使用User与UserWordNeighbors的关联度，来表征用户；

当然计算的时候，是对User的每个UserWord都执行以上过程；
UserWord的权重可以对UserWord出现的数量取-1*logn；

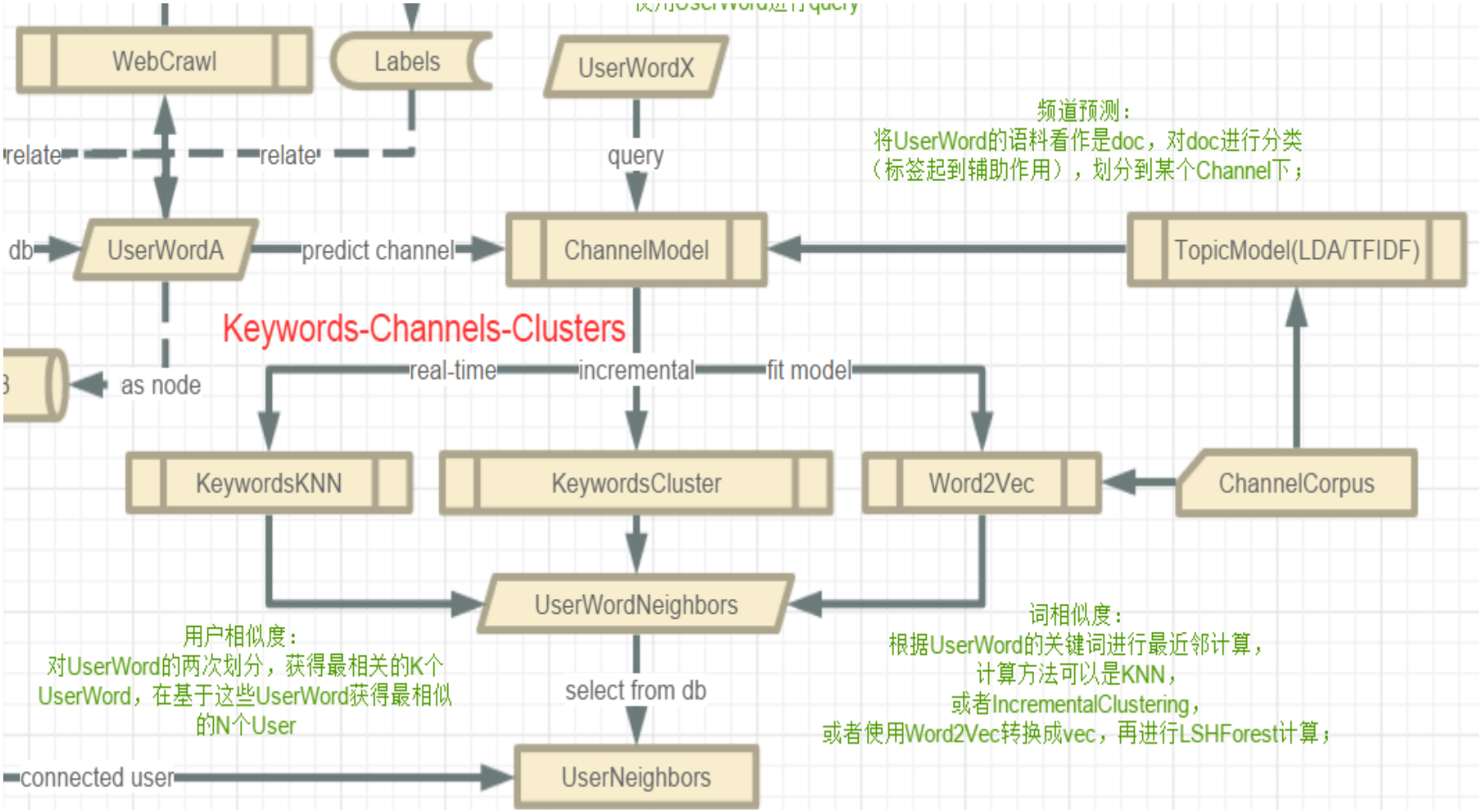
以下是改进模型的UserWord关键词提取；
除了Keywords，我们还会提取Labels，对分类起到精度提升；
UserWordCorpus指的是和UserWord相关的内容；

词条标签： 漫画，动漫形象，动漫，其他，人物

当然下图还提到了，对同意异形，一词多义，一词歧义；
因为这些都是非常重要且复杂的处理，所以这里不展开讨论；



以下是改进模型的UserWord最近邻模型；
其中分类算法，可以使用RocchioClassifier，也可以使用LDA，选择取决于语料库是否具有类别标签；
当然如果没有语料库的话，也可以人工向每个类别添加关键词（相当于是对简化版的RocchioClassifier，也是有效的）；



基于关键词的聚类，比较复杂的方法是训练一个Word2Vec模型，然后使用训练样本的词向量构建一个LSHForest(Local Sensitive Hashing Forest)；两者都可以增量训练；

但是这个方法因为复杂度和计算量都比较高，因此我们考虑用更简单的聚类树来做，伪代码由下图给出；

增量式关键词聚类伪代码－随机KMeans森林：

划分过程：
随机选择两个没有共有词的质心；
对其余的每个数据点做：
　　划分到其中一个质心，如果和任意一个质心都没有共有词，则将这个点添加为新的质心；

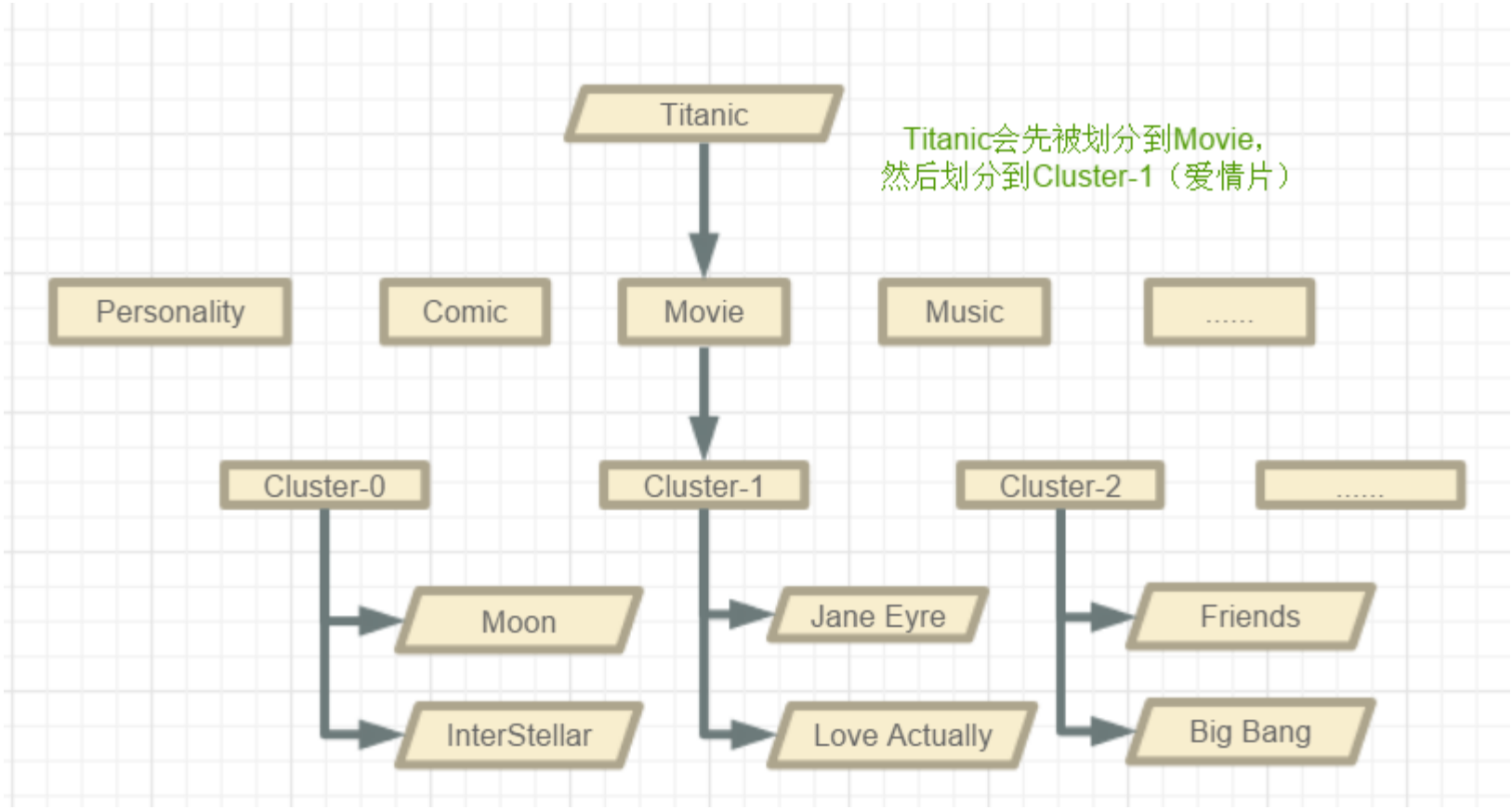
迭代过程：
对每个划分好的簇做：
　　如果簇size符合要求，则停止对簇划分；
　　否则的话，对簇执行划分过程；

树增长过程：
对新的数据点，从根节点开始，不断划分，直到被划分到叶子节点；
对叶子节点做判断，如果簇size符合要求，则对簇进行划分；

可以同时生成多个树，对结果加权聚合，以提高聚类精度；

其中的重点在于，
质心的选择需要尽可能保证彼此互不相似；

下图是一个效果展示，所有的词都是UserWord；
第一层分类，是大范围划分，在实现上很可能会采取分布式；
第二层分类，因为像科幻片，会出现共有关键词可能性比较高，比如星球，太空等等，而爱情片，则会出现爱情，伴侣等词汇；



用户最近邻计算由以下伪代码给出：

用户最近邻伪代码：
将用户自己的所有用户词，作为关联词，权值设为10；
将用户自己的所有用户词所属于的簇，作为关联簇，权值设为1；

统计所有关联词和关联簇所连接的其他用户的出现次数，并加权求和；

我们将这个模型称为KCC，Keywords-Channels-Clusters；

事实上，只要某种实体具有相关描述的document，我们就可以使用KCC模型计算这种实体的相似度，不管是UserWord还是UGC，甚至是News；

也就是说，我们可以用同一套KCC，同时使用UserWord/UGC/News等作为模型数据，进行模型训练，这样一来，可以将UserWord/UGC/News等进行关联；

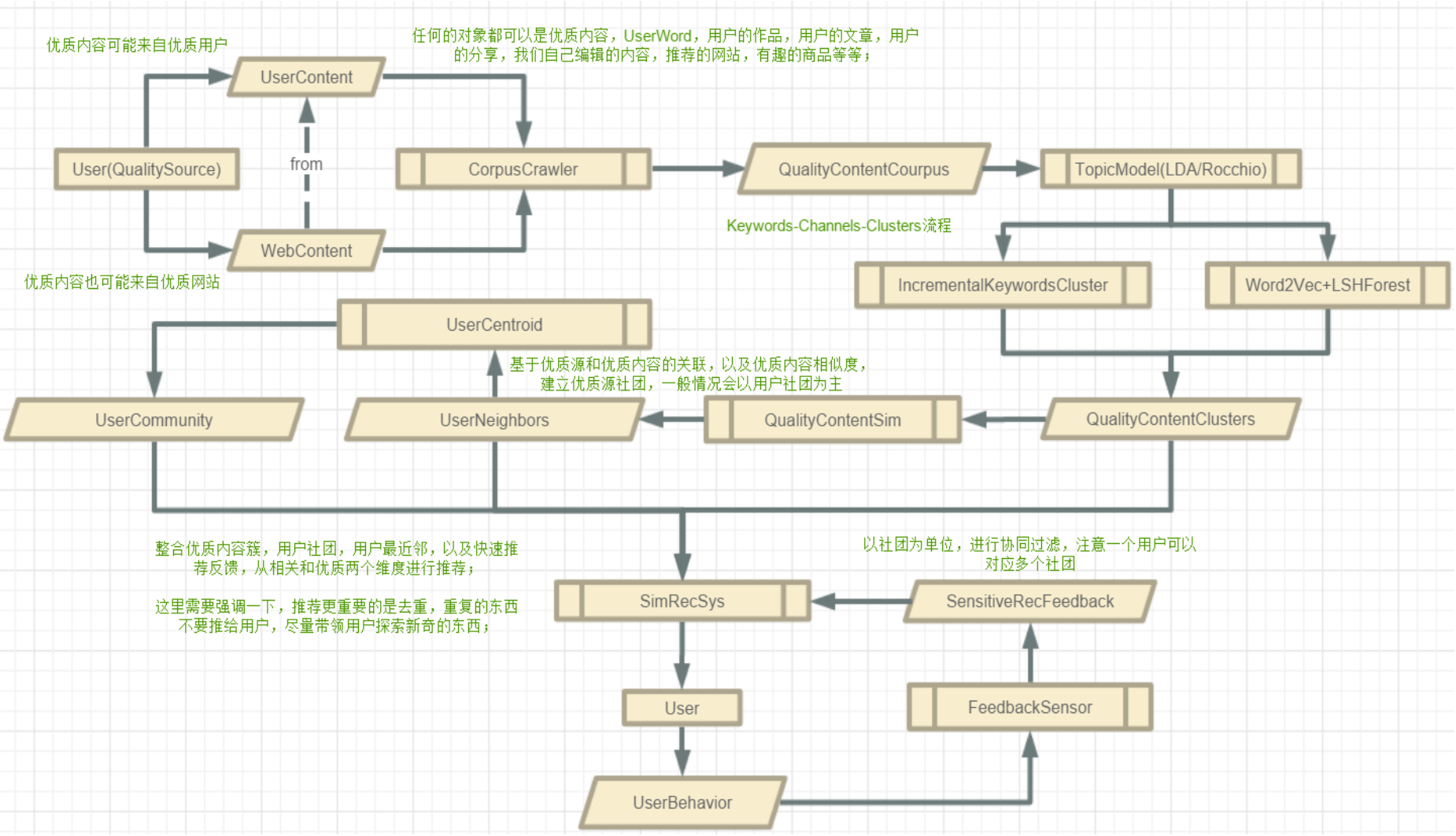
提供User相似度的K-UserCentroids模型

KCC解决了Item-Based的问题，我们还需要建立对User的协同过滤（User-Based）；

我们的计算策略是，以活跃度高的若干个不相似的K个User作为质心，再将剩余的User划分到最相似的TopN-UserCentroids，同时会保留User对这K-UserCentroids的相似度（K个是全部，N个是Top），作为推荐计算的权重；

当然，K-UserCentroids的选择，还可以融合用户的行为数据，提升质心选择的质量

最后，我们基于KCC和K-UserCentroids构建SimRecSys，
其核心功能是，根据兴趣相关(KCC)和协同过滤(K-UserCentroids)，为用户过滤去那些他很可能不感兴趣的Content或User（注意这里用的是过滤，而非推荐）；
以下是总架构图；



以上模型中有很多细节目前还未进行精化，同时也有很多需要加强的地方，另外也还没能对模型进行实验验证；