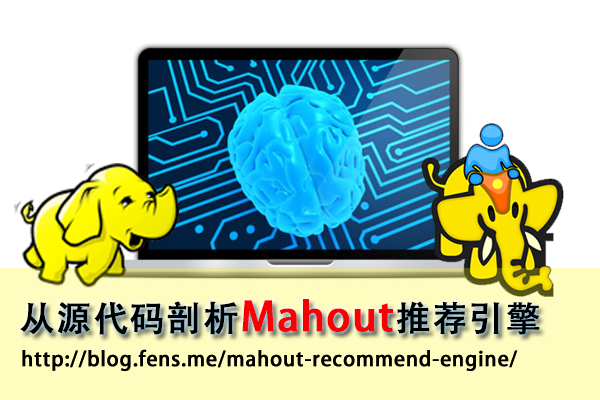
[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-recommender-engine.png)

**前言**

Mahout框架中cf.taste包实现了推荐算法引擎，它提供了一套完整的推荐算法工具集，同时规范了数据结构，并标准化了程序开发过程。应用推荐算法时，代码也就7-8行，简单地有点像R了。为了使用简单的目标，Mahout推荐引擎必然要做到精巧的程序设计。

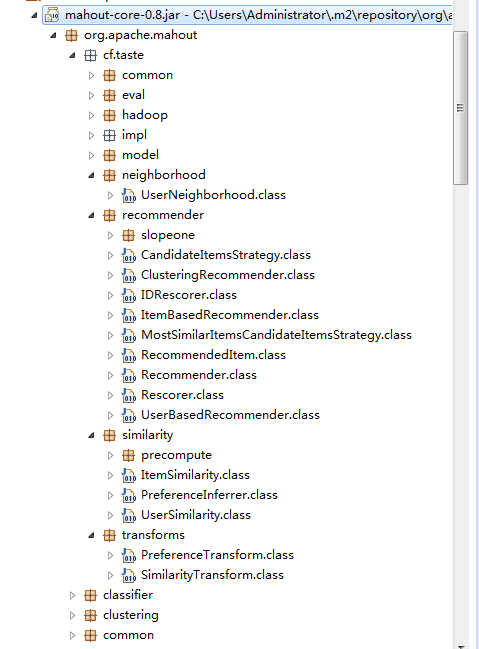
本文将介绍Mahout推荐引擎的程序设计。

**目录**

1. Mahout推荐引擎概况
2. 标准化的程序开发过程
3. 数据模型
4. 相似度算法工具集
5. 近邻算法工具集
6. 推荐算法工具集
7. 创建自己的推荐引擎构造器

**1. Mahout推荐引擎概况**

Mahout的推荐引擎，要从org.apache.mahout.cf.taste包说起。

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-core-class.png)

packages的说明：

* common: 公共类包括，异常，数据刷新接口，权重常量
* eval: 定义构造器接口，类似于工厂模式
* model: 定义数据模型接口
* neighborhood: 定义近邻算法的接口
* recommender: 定义推荐算法的接口
* similarity: 定义相似度算法的接口
* transforms: 定义数据转换的接口
* hadoop: 基于hadoop的分步式算法的实现类
* impl: 单机内存算法实现类

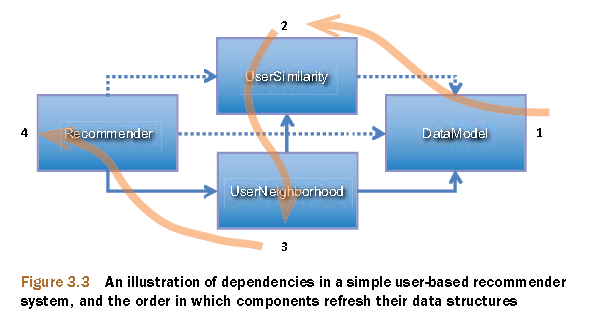
从上面的package情况，我可以粗略地看出推荐引擎分为5个主要部分组成：数据模型，相似度算法，近邻算法，推荐算法，算法评分器。

从数据处理能力上，算法可以分为：单机内存算法，基于hadoop的分步式算法。

下面我们将基于单机内存算法，研究Mahout的推荐引擎的结构。

**2. 标准化的程序开发过程**

以UserCF的推荐算法为例，官方建议我们的开发过程：

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout_recommendation-process.png)

图片摘自Mahout in Action

从上图中我们可以看到，算法是被模块化的，通过1,2,3,4的过程进行方法调用。

程序代码：

public class UserCF {

final static int NEIGHBORHOOD\_NUM = 2;

final static int RECOMMENDER\_NUM = 3;

public static void main(String[] args) throws IOException, TasteException {

String file = "datafile/item.csv";

DataModel model = new FileDataModel(new File(file));

UserSimilarity user = new EuclideanDistanceSimilarity(model);

NearestNUserNeighborhood neighbor = new NearestNUserNeighborhood(NEIGHBORHOOD\_NUM, user, model);

Recommender r = new GenericUserBasedRecommender(model, neighbor, user);

LongPrimitiveIterator iter = model.getUserIDs();

while (iter.hasNext()) {

long uid = iter.nextLong();

List list = r.recommend(uid, RECOMMENDER\_NUM);

System.out.printf("uid:%s", uid);

for (RecommendedItem ritem : list) {

System.out.printf("(%s,%f)", ritem.getItemID(), ritem.getValue());

}

System.out.println();

}

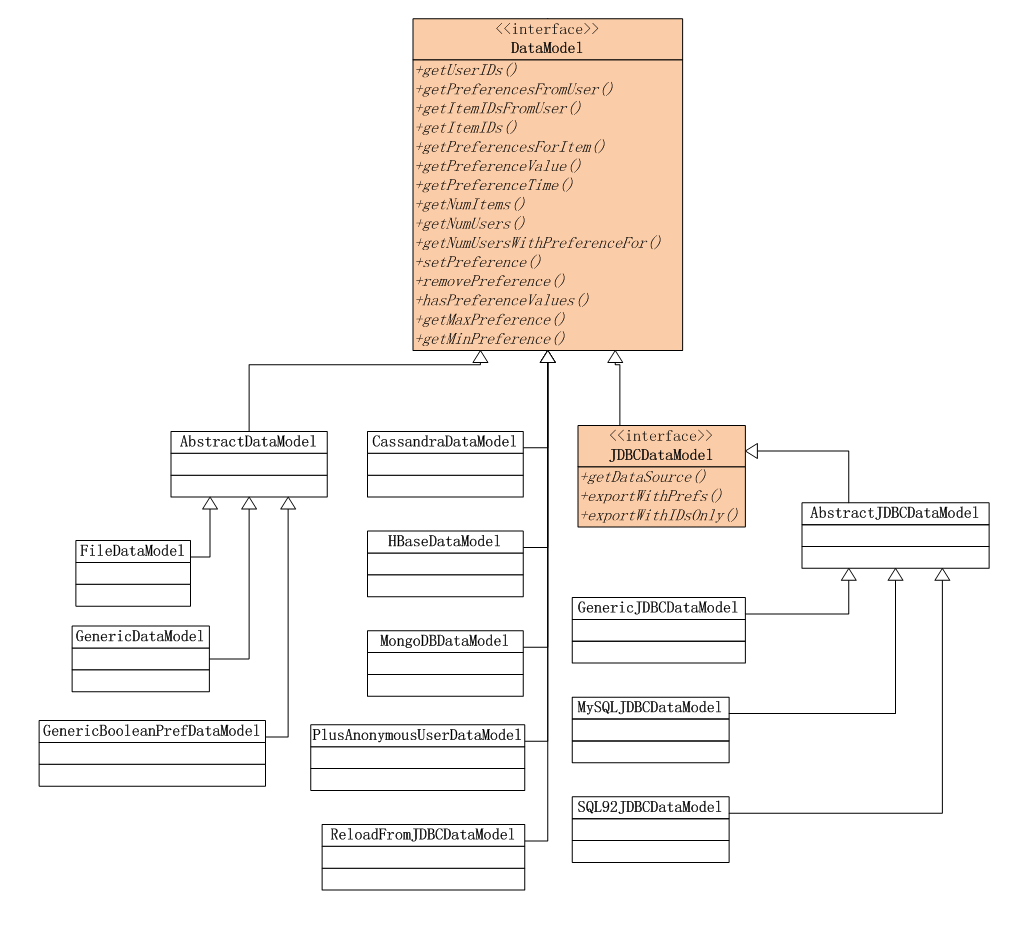
}

}

我们调用算法的程序，要用到4个对象：DataModel, UserSimilarity, NearestNUserNeighborhood, Recommender。

**3. 数据模型**

Mahout的推荐引擎的数据模型，以DataModel接口为父类。

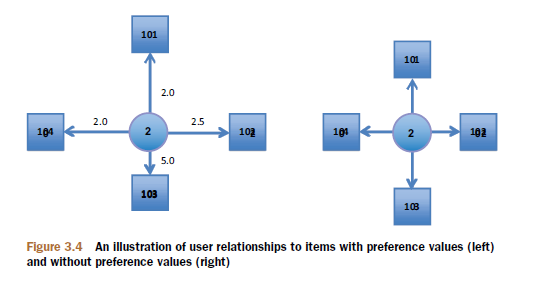
[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-datamodel.png)

通过“策略模式”匹配不同的数据源，支持File, JDBC(MySQL, PostgreSQL), NoSQL(Cassandra, HBase, MongoDB)。

注：NoSQL的实现在mahout-integration-0.8.jar中。

数据格式支持2种：

* GenericDataModel: 用户ID，物品ID，用户对物品的打分(UserID,ItemID,PreferenceValue)
* GenericBooleanPrefDataModel: 用户ID，物品ID (UserID,ItemID)，这种方式表达用户是否浏览过该物品，但并未对物品进行打分。

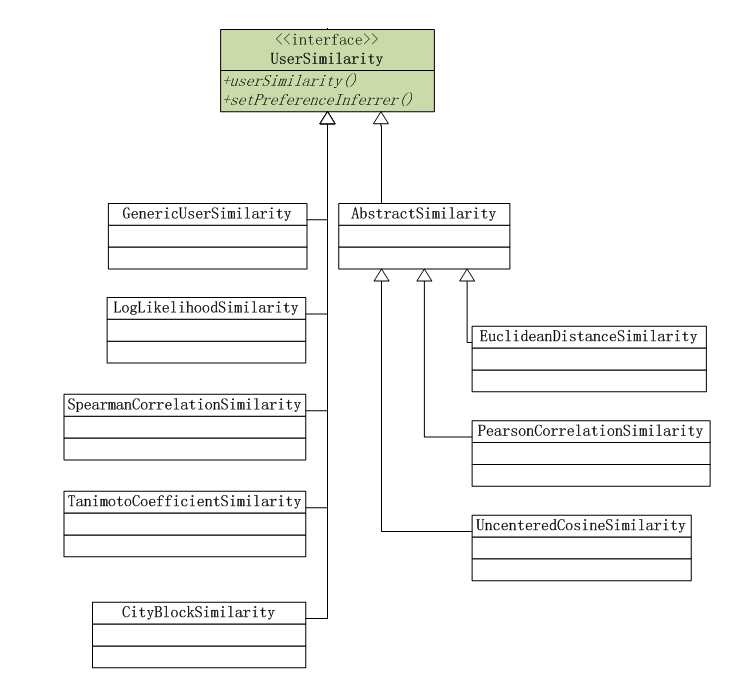
[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-pref.png)

**4. 相似度算法工具集**

相似度算法分为2种

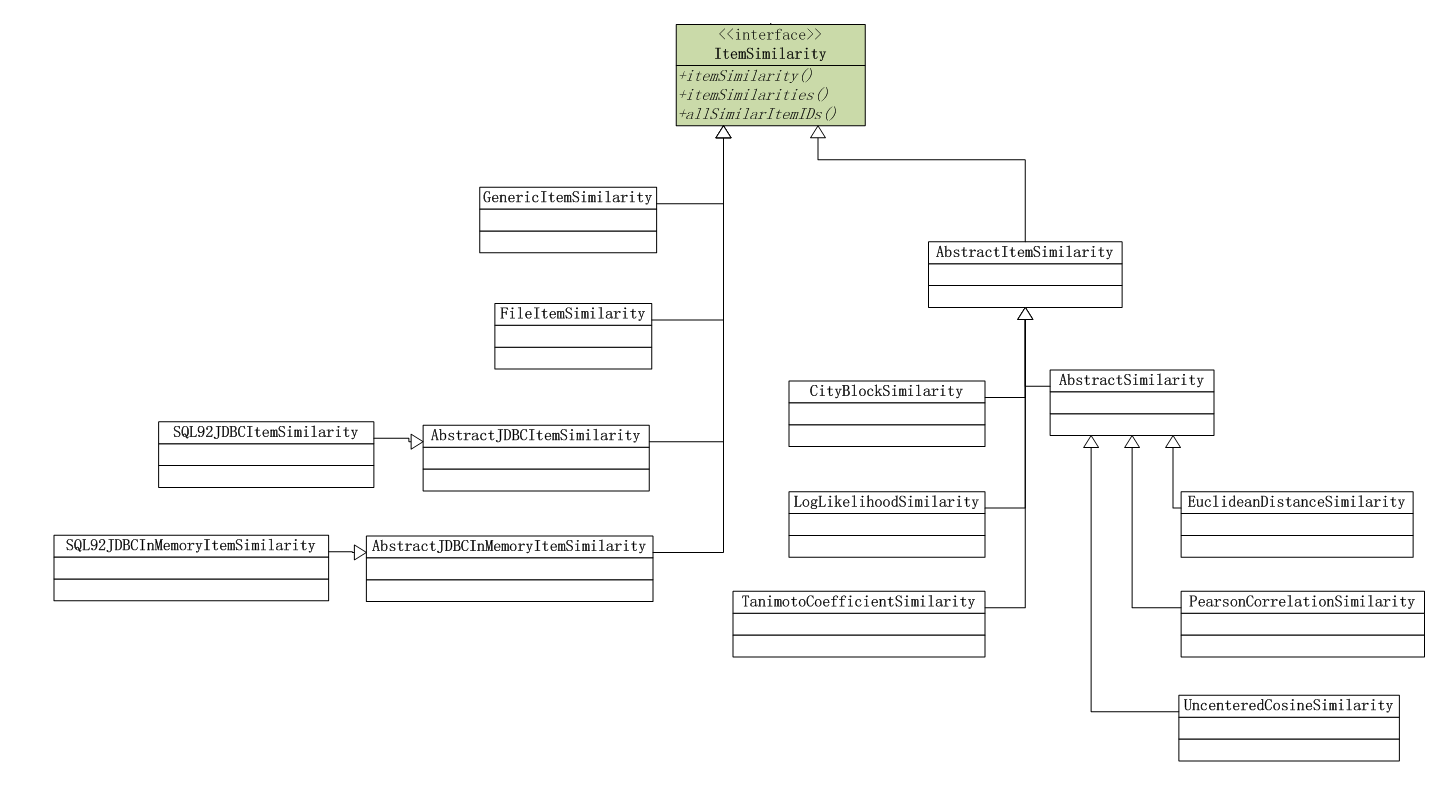
* 基于用户(UserCF)的相似度算法
* 基于物品(ItemCF)的相似度算法

**1). 基于用户(UserCF)的相似度算法**

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-UserSimilarity.png)

计算用户的相似矩阵，可以通过上图中几种算法。

**2). 基于物品(ItemCF)的相似度算法**

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-ItemSimilarity.png)

计算物品的相似矩阵，可以通过上图中几种算法。

关于相似度距离的说明：

* **EuclideanDistanceSimilarity: 欧氏距离相似度**

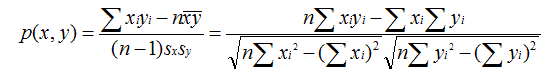
[image003](http://blog.fens.me/wp-content/uploads/2013/10/image003.gif)

原理：利用欧式距离d定义的相似度s，s=1 / (1+d)。

范围：[0,1]，值越大，说明d越小，也就是距离越近，则相似度越大。

说明：同皮尔森相似度一样，该相似度也没有考虑重叠数对结果的影响，同样地，Mahout通过增加一个枚举类型（Weighting）的参数来使得重叠数也成为计算相似度的影响因子。

* **PearsonCorrelationSimilarity: 皮尔森相似度**

[](http://blog.fens.me/wp-content/uploads/2013/10/image004.gif)

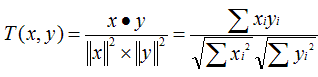
原理：用来反映两个变量线性相关程度的统计量

范围：[-1,1]，绝对值越大，说明相关性越强，负相关对于推荐的意义小。

说明：1、 不考虑重叠的数量；2、 如果只有一项重叠，无法计算相似性（计算过程被除数有n-1）；3、 如果重叠的值都相等，也无法计算相似性（标准差为0，做除数）。

该相似度并不是最好的选择，也不是最坏的选择，只是因为其容易理解，在早期研究中经常被提起。使用Pearson线性相关系数必须假设数据是成对地从正态分布中取得的，并且数据至少在逻辑范畴内必须是等间距的数据。Mahout中，为皮尔森相关计算提供了一个扩展，通过增加一个枚举类型（Weighting）的参数来使得重叠数也成为计算相似度的影响因子。

* **UncenteredCosineSimilarity: 余弦相似度**

[](http://blog.fens.me/wp-content/uploads/2013/10/image005.gif)

原理：多维空间两点与所设定的点形成夹角的余弦值。

范围：[-1,1]，值越大，说明夹角越大，两点相距就越远，相似度就越小。

说明：在数学表达中，如果对两个项的属性进行了数据中心化，计算出来的余弦相似度和皮尔森相似度是一样的，在mahout中，实现了数据中心化的过程，所以皮尔森相似度值也是数据中心化后的余弦相似度。另外在新版本中，Mahout提供了UncenteredCosineSimilarity类作为计算非中心化数据的余弦相似度。

* **SpearmanCorrelationSimilarity: Spearman秩相关系数相似度**

原理：Spearman秩相关系数通常被认为是排列后的变量之间的Pearson线性相关系数。

范围：{-1.0,1.0}，当一致时为1.0，不一致时为-1.0。

说明：计算非常慢，有大量排序。针对推荐系统中的数据集来讲，用Spearman秩相关系数作为相似度量是不合适的。

* **CityBlockSimilarity: 曼哈顿距离相似度**

原理：曼哈顿距离的实现，同欧式距离相似，都是用于多维数据空间距离的测度

范围：[0,1]，同欧式距离一致，值越小，说明距离值越大，相似度越大。

说明：比欧式距离计算量少，性能相对高。

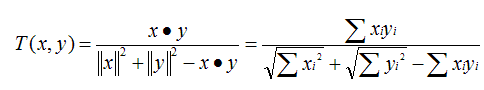
* **LogLikelihoodSimilarity: 对数似然相似度**

原理：重叠的个数，不重叠的个数，都没有的个数

范围：具体可去百度文库中查找论文《Accurate Methods for the Statistics of Surprise and Coincidence》

说明：处理无打分的偏好数据，比Tanimoto系数的计算方法更为智能。

* **TanimotoCoefficientSimilarity: Tanimoto系数相似度**

[](http://blog.fens.me/wp-content/uploads/2013/10/image006.gif)

原理：又名广义Jaccard系数，是对Jaccard系数的扩展，等式为

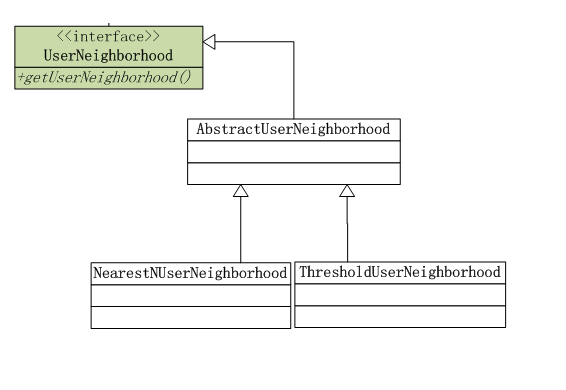
范围：[0,1]，完全重叠时为1，无重叠项时为0，越接近1说明越相似。

说明：处理无打分的偏好数据。

相似度算法介绍，摘自：<http://www.cnblogs.com/dlts26/archive/2012/06/20/2555772.html>

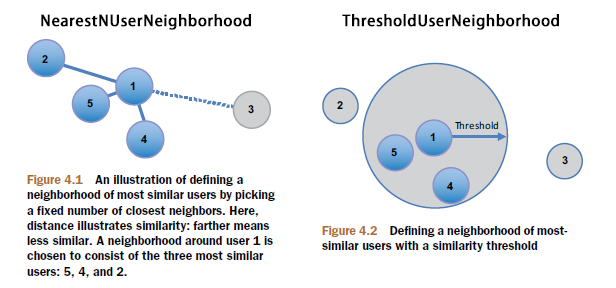
**5. 近邻算法工具集**

近邻算法只对于UserCF适用，通过近邻算法给相似的用户进行排序，选出前N个最相似的，作为最终推荐的参考的用户。

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-UserNeighborhood.png)

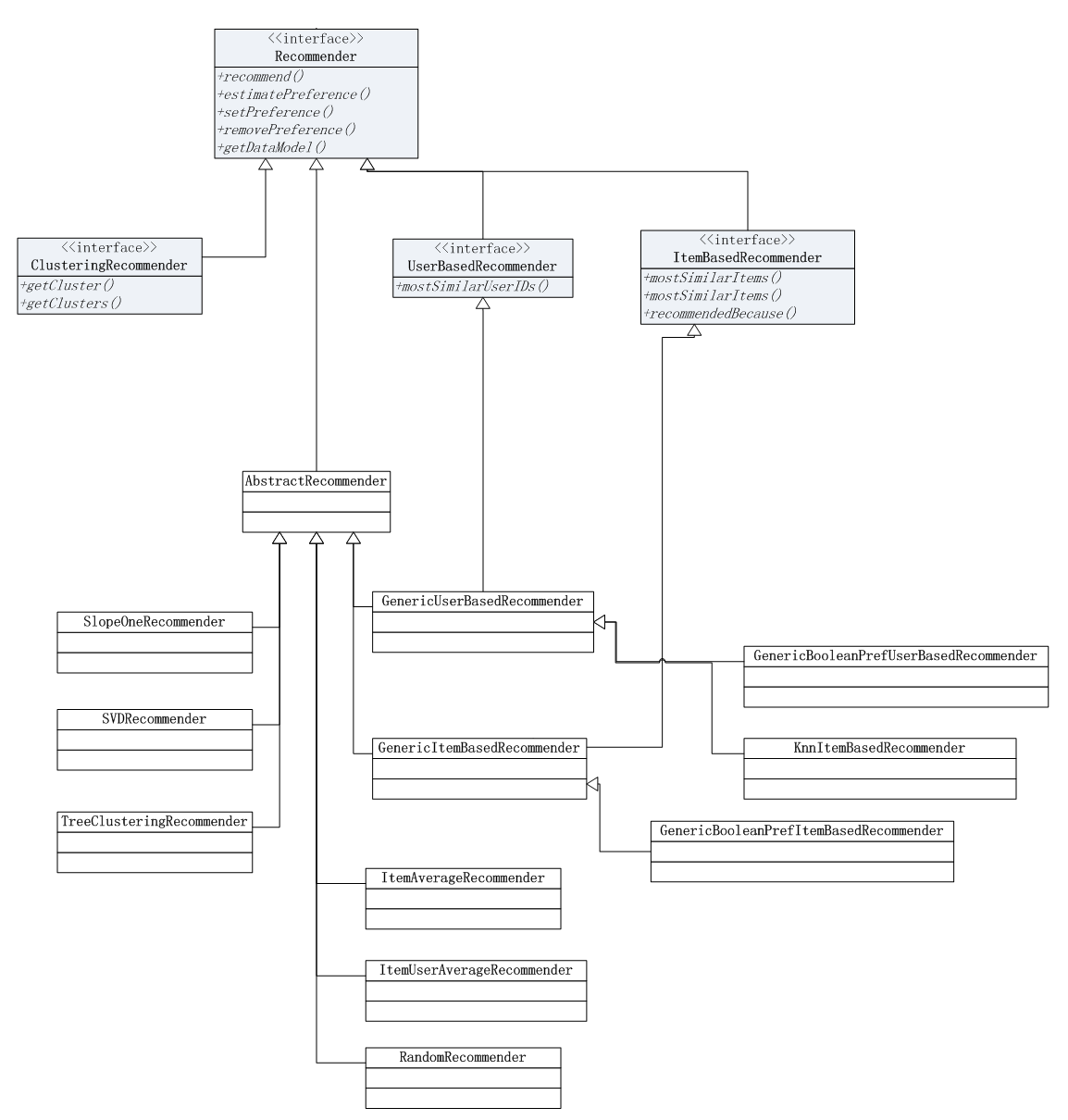
近邻算法分为2种：

* NearestNUserNeighborhood:指定N的个数，比如，选出前10最相似的用户。
* ThresholdUserNeighborhood:指定比例，比如，选择前10%最相似的用户。

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-Neighborhood.png)

**6. 推荐算法工具集**

推荐算法是以Recommender作为基础的父类，关于推荐算法的详细介绍，请参考文章：[Mahout推荐算法API详解](http://blog.fens.me/mahout-recommendation-api)

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-Recommender.png)

**7. 创建自己的推荐引擎构造器**

有了上面的知识，我就清楚地知道了Mahout推荐引擎的原理和使用，我们就可以写一个自己的构造器，通过“策略模式”实现，算法的组合。

新建文件：org.conan.mymahout.recommendation.job.RecommendFactory.java

public final class RecommendFactory {

...

}

**1). 构造数据模型**

public static DataModel buildDataModel(String file) throws TasteException, IOException {

return new FileDataModel(new File(file));

}

public static DataModel buildDataModelNoPref(String file) throws TasteException, IOException {

return new GenericBooleanPrefDataModel(GenericBooleanPrefDataModel.toDataMap(new FileDataModel(new File(file))));

}

public static DataModelBuilder buildDataModelNoPrefBuilder() {

return new DataModelBuilder() {

@Override

public DataModel buildDataModel(FastByIDMap trainingData) {

return new GenericBooleanPrefDataModel(GenericBooleanPrefDataModel.toDataMap(trainingData));

}

};

}

**2). 构造相似度算法模型**

public enum SIMILARITY {

PEARSON, EUCLIDEAN, COSINE, TANIMOTO, LOGLIKELIHOOD, FARTHEST\_NEIGHBOR\_CLUSTER, NEAREST\_NEIGHBOR\_CLUSTER

}

public static UserSimilarity userSimilarity(SIMILARITY type, DataModel m) throws TasteException {

switch (type) {

case PEARSON:

return new PearsonCorrelationSimilarity(m);

case COSINE:

return new UncenteredCosineSimilarity(m);

case TANIMOTO:

return new TanimotoCoefficientSimilarity(m);

case LOGLIKELIHOOD:

return new LogLikelihoodSimilarity(m);

case EUCLIDEAN:

default:

return new EuclideanDistanceSimilarity(m);

}

}

public static ItemSimilarity itemSimilarity(SIMILARITY type, DataModel m) throws TasteException {

switch (type) {

case LOGLIKELIHOOD:

return new LogLikelihoodSimilarity(m);

case TANIMOTO:

default:

return new TanimotoCoefficientSimilarity(m);

}

}

public static ClusterSimilarity clusterSimilarity(SIMILARITY type, UserSimilarity us) throws TasteException {

switch (type) {

case NEAREST\_NEIGHBOR\_CLUSTER:

return new NearestNeighborClusterSimilarity(us);

case FARTHEST\_NEIGHBOR\_CLUSTER:

default:

return new FarthestNeighborClusterSimilarity(us);

}

}

**3). 构造近邻算法模型**

public enum NEIGHBORHOOD {

NEAREST, THRESHOLD

}

public static UserNeighborhood userNeighborhood(NEIGHBORHOOD type, UserSimilarity s, DataModel m, double num) throws TasteException {

switch (type) {

case NEAREST:

return new NearestNUserNeighborhood((int) num, s, m);

case THRESHOLD:

default:

return new ThresholdUserNeighborhood(num, s, m);

}

}

**4). 构造推荐算法模型**

public enum RECOMMENDER {

USER, ITEM

}

public static RecommenderBuilder userRecommender(final UserSimilarity us, final UserNeighborhood un, boolean pref) throws TasteException {

return pref ? new RecommenderBuilder() {

@Override

public Recommender buildRecommender(DataModel model) throws TasteException {

return new GenericUserBasedRecommender(model, un, us);

}

} : new RecommenderBuilder() {

@Override

public Recommender buildRecommender(DataModel model) throws TasteException {

return new GenericBooleanPrefUserBasedRecommender(model, un, us);

}

};

}

public static RecommenderBuilder itemRecommender(final ItemSimilarity is, boolean pref) throws TasteException {

return pref ? new RecommenderBuilder() {

@Override

public Recommender buildRecommender(DataModel model) throws TasteException {

return new GenericItemBasedRecommender(model, is);

}

} : new RecommenderBuilder() {

@Override

public Recommender buildRecommender(DataModel model) throws TasteException {

return new GenericBooleanPrefItemBasedRecommender(model, is);

}

};

}

public static RecommenderBuilder slopeOneRecommender() throws TasteException {

return new RecommenderBuilder() {

@Override

public Recommender buildRecommender(DataModel dataModel) throws TasteException {

return new SlopeOneRecommender(dataModel);

}

};

}

public static RecommenderBuilder itemKNNRecommender(final ItemSimilarity is, final Optimizer op, final int n) throws TasteException {

return new RecommenderBuilder() {

@Override

public Recommender buildRecommender(DataModel dataModel) throws TasteException {

return new KnnItemBasedRecommender(dataModel, is, op, n);

}

};

}

public static RecommenderBuilder svdRecommender(final Factorizer factorizer) throws TasteException {

return new RecommenderBuilder() {

@Override

public Recommender buildRecommender(DataModel dataModel) throws TasteException {

return new SVDRecommender(dataModel, factorizer);

}

};

}

public static RecommenderBuilder treeClusterRecommender(final ClusterSimilarity cs, final int n) throws TasteException {

return new RecommenderBuilder() {

@Override

public Recommender buildRecommender(DataModel dataModel) throws TasteException {

return new TreeClusteringRecommender(dataModel, cs, n);

}

};

}

**5). 构造算法评估模型**

public enum EVALUATOR {

AVERAGE\_ABSOLUTE\_DIFFERENCE, RMS

}

public static RecommenderEvaluator buildEvaluator(EVALUATOR type) {

switch (type) {

case RMS:

return new RMSRecommenderEvaluator();

case AVERAGE\_ABSOLUTE\_DIFFERENCE:

default:

return new AverageAbsoluteDifferenceRecommenderEvaluator();

}

}

public static void evaluate(EVALUATOR type, RecommenderBuilder rb, DataModelBuilder mb, DataModel dm, double trainPt) throws TasteException {

System.out.printf("%s Evaluater Score:%s\n", type.toString(), buildEvaluator(type).evaluate(rb, mb, dm, trainPt, 1.0));

}

public static void evaluate(RecommenderEvaluator re, RecommenderBuilder rb, DataModelBuilder mb, DataModel dm, double trainPt) throws TasteException {

System.out.printf("Evaluater Score:%s\n", re.evaluate(rb, mb, dm, trainPt, 1.0));

}

/\*\*

\* statsEvaluator

\*/

public static void statsEvaluator(RecommenderBuilder rb, DataModelBuilder mb, DataModel m, int topn) throws TasteException {

RecommenderIRStatsEvaluator evaluator = new GenericRecommenderIRStatsEvaluator();

IRStatistics stats = evaluator.evaluate(rb, mb, m, null, topn, GenericRecommenderIRStatsEvaluator.CHOOSE\_THRESHOLD, 1.0);

// System.out.printf("Recommender IR Evaluator: %s\n", stats);

System.out.printf("Recommender IR Evaluator: [Precision:%s,Recall:%s]\n", stats.getPrecision(), stats.getRecall());

}

**6). 推荐结果输出**

public static void showItems(long uid, List recommendations, boolean skip) {

if (!skip || recommendations.size() > 0) {

System.out.printf("uid:%s,", uid);

for (RecommendedItem recommendation : recommendations) {

System.out.printf("(%s,%f)", recommendation.getItemID(), recommendation.getValue());

}

System.out.println();

}

}

**7). 完整源代码文件及使用样例：**  
<https://github.com/bsspirit/maven_mahout_template/tree/mahout-0.8/src/main/java/org/conan/mymahout/recommendation/job>