基于用户的推荐

产生一些样本数据

```
idList[Item] = Range[10];
idList[User] = CharacterRange["A", "Z"];

nameList[Item] = StringTemplate["Item-`1`"] /@ idList[Item];

nameList[User] = StringTemplate["User-`1`"] /@ idList[User];

asso[Item] = AssociationThread[nameList[Item] -> idList[Item]]
```

```
<|Item-1 → 1, Item-2 → 2, Item-3 → 3, Item-4 → 4, Item-5 → 5,
 Item-6 → 6, Item-7 → 7, Item-8 → 8, Item-9 → 9, Item-10 → 10|>
```

```
asso[User] = AssociationThread[nameList[User] -> idList[User]]
```

```
<|User-A → A, User-B → B, User-C → C, User-D → D, User-E → E, User-F → F, User-G → G,
 User-H → H, User-I → I, User-J → J, User-K → K, User-L → L, User-M → M, User-N → N,
 User-O → O, User-P → P, User-Q → Q, User-R → R, User-S → S, User-T → T,
 User-U → U, User-V → V, User-W → W, User-X → X, User-Y → Y, User-Z → Z|>
```

```
dims = Length /@ {nameList[User], nameList[Item]}
```

```
{26, 10}
```

产生一每个Item的值，出现的频次

```
KeySort@Counts@RandomChoice[idList[Item], 50]
```

```
<|1 → 6, 2 → 5, 3 → 5, 4 → 6, 6 → 4, 7 → 6, 8 → 6, 9 → 8, 10 → 4|>
```

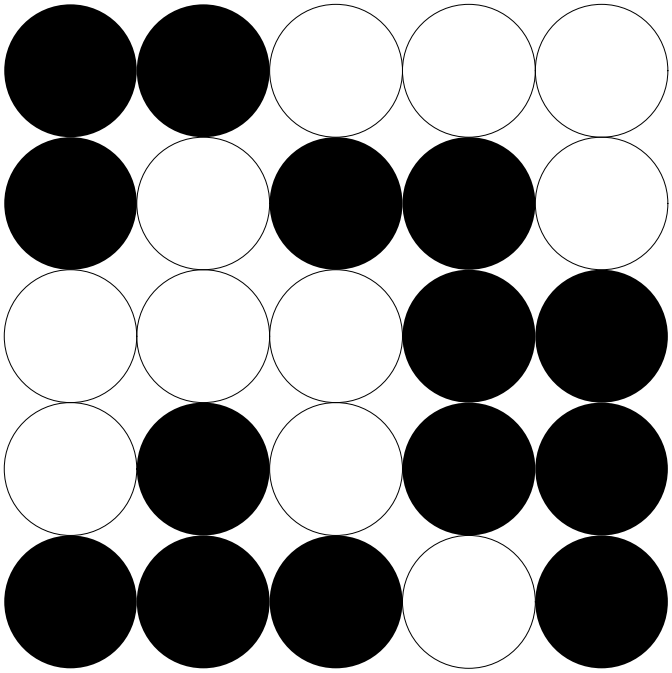
这种方式，所有的Item都有值，实际情形中，是非常稀疏的，不合理。

```
RandomInteger[{1, 5}] # & /@ (KeySort@Counts@RandomChoice[idList[Item], 6])
```

```
<|1 → 5, 4 → 2, 5 → 5, 8 → 4, 10 → 4|>
```

那么现在直接产生这样一个矩阵，从一个矩阵里随机取一些位置[分布不均匀或有明显规则]，并赋值一些评分。
暂时也可以用一个固定规则产生一个固定的看起来杂乱随机的值。

```
Graphics[Table[RandomChoice[{Disk, Circle}][2 {i, j}], {i, 5}, {j, 5}]]
```



```
asso[Data] = KeySort[Counts[#]] & /@ RandomInteger[{2, 10}, dims];

AssociationPad[asso_, keysFull_] :=
 AssociationThread[keysFull -> Lookup[asso, keysFull, 0]]

assoValue[User] = AssociationPad[#, Range[10]] & /@ asso[Data];
```

转换成一个表格

```
grid = Values /@ assoValue[User]
```

```
{{0, 1, 1, 0, 2, 2, 3, 0, 0, 1}, {0, 1, 0, 0, 3, 0, 1, 2, 1, 2},
 {0, 0, 0, 2, 1, 1, 2, 3, 0, 1}, {0, 2, 0, 0, 3, 0, 2, 0, 2, 1},
 {0, 1, 2, 1, 1, 1, 0, 1, 1, 2}, {0, 0, 1, 2, 0, 0, 1, 4, 0, 2},
 {0, 0, 0, 1, 1, 0, 1, 4, 2, 1}, {0, 1, 1, 0, 2, 3, 1, 0, 2, 0},
 {0, 0, 0, 3, 1, 0, 2, 1, 3, 0}, {0, 2, 1, 1, 1, 0, 0, 3, 0, 2},
 {0, 2, 1, 0, 1, 2, 2, 1, 0, 1}, {0, 0, 3, 0, 3, 2, 1, 1, 0, 0},
 {0, 1, 2, 2, 0, 1, 0, 1, 1, 2}, {0, 2, 2, 1, 1, 1, 1, 0, 0, 2},
 {0, 3, 1, 3, 0, 0, 1, 1, 0, 1}, {0, 2, 1, 0, 0, 2, 1, 2, 1, 1},
 {0, 2, 1, 0, 3, 2, 0, 1, 1, 0}, {0, 0, 1, 1, 2, 1, 2, 0, 1, 2},
 {0, 3, 1, 1, 0, 3, 0, 1, 0, 1}, {0, 1, 1, 2, 1, 1, 1, 2, 1, 0},
 {0, 1, 0, 3, 0, 1, 4, 0, 0, 1}, {0, 0, 0, 0, 4, 0, 2, 2, 2, 0},
 {0, 1, 2, 1, 0, 1, 1, 3, 1, 0}, {0, 0, 2, 4, 0, 1, 2, 0, 0, 1},
 {0, 0, 3, 0, 0, 3, 1, 2, 1, 0}, {0, 0, 3, 0, 0, 0, 0, 3, 2, 2}}
```

```
TableForm[grid, TableHeadings → {nameList[User], nameList[Item]}]
```

	Item-1	Item-2	Item-3	Item-4	Item-5	Item-6	Item-7	Item
User-A	0	1	1	0	2	2	3	0
User-B	0	1	0	0	3	0	1	2
User-C	0	0	0	2	1	1	2	3
User-D	0	2	0	0	3	0	2	0
User-E	0	1	2	1	1	1	0	1
User-F	0	0	1	2	0	0	1	4
User-G	0	0	0	1	1	0	1	4
User-H	0	1	1	0	2	3	1	0
User-I	0	0	0	3	1	0	2	1
User-J	0	2	1	1	1	0	0	3
User-K	0	2	1	0	1	2	2	1
User-L	0	0	3	0	3	2	1	1
User-M	0	1	2	2	0	1	0	1
User-N	0	2	2	1	1	1	1	0
User-O	0	3	1	3	0	0	1	1
User-P	0	2	1	0	0	2	1	2
User-Q	0	2	1	0	3	2	0	1
User-R	0	0	1	1	2	1	2	0
User-S	0	3	1	1	0	3	0	1
User-T	0	1	1	2	1	1	1	2
User-U	0	1	0	3	0	1	4	0
User-V	0	0	0	0	4	0	2	2
User-W	0	1	2	1	0	1	1	3
User-X	0	0	2	4	0	1	2	0
User-Y	0	0	3	0	0	3	1	2
User-Z	0	0	3	0	0	0	0	3

综合评分模块

那么许多时候，一个简单的推荐系统，重点在于如何产生一个评分矩阵，因为在实际中，比如当当网这种，自己的网站，可能有用户自己的评分，但是许多时候用户的评分[比如淘宝大家都给好评多]并不准确，那么涉及到一个评分的处理，而有的场景，则没有实际的评分，那么就可以通过构造[评分]，比如用户的偏好，比如用户浏览过的频次，及各种综合起来后的[评分]。

从购买相关行为的维度来看，我们可以制定

```
actionList = {购买, 浏览, 收藏, 购物车};
```

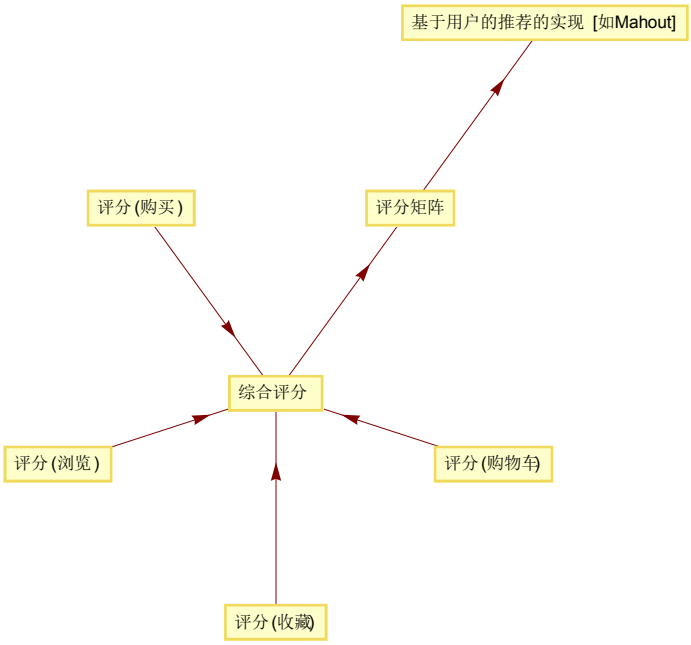
```
list[评分] = 评分/@actionList
```

```
{评分[购买], 评分[浏览], 评分[收藏], 评分[购物车]}
```

```
rules = {Thread[list[评分] → 综合评分], 综合评分→ 评分矩阵,
    评分矩阵→ "基于用户的推荐的实现[如Mahout]"} // Flatten
```

```
{评分[购买] → 综合评分, 评分[浏览] → 综合评分, 评分[收藏] → 综合评分,
 评分[购物车] → 综合评分, 综合评分→ 评分矩阵, 评分矩阵→ 基于用户的推荐的实现[如Mahout]}
```

```
GraphPlot[Thread /@ rules, VertexLabeling → True, DirectedEdges → True]
```



可能的问题

许多时候，一个统一的模型要应对不同的商家类目，参数情况是不一样的。

比如购买评分，如果复购率较高的店铺，则购买评分的权重高是有正面效果的，相反，如果是大件，比如半年内买过一次空调的，可能压根就不会再购买，再怎么推荐也没有用，但是可以推荐相关的/关联的物品，因此许多混合算法中，会剔除购买过的物品。

新商品问题，新的商品没有历史用户的偏好与评分，则可以找一些相似的商品进行预评分

计算评分矩阵[基于用户的协同过滤算法]

在样例评分矩阵中，分数为0的要补全，补全后，即可以对每个用户按评分排序，而已经有的评分物品中，也是可以剔除的。
但这个不是现在的重点，得具体看混合算法或改进中的实现。

计算用户的相似度[查找用户的近邻]

皮尔逊相关系数的计算公式，两个用户A，B

$$s(A, B) = \frac{\sum_{i \in I_A \cap I_B} (r_{A,i} - \text{Mean}[r_A])(r_{B,i} - \text{Mean}[r_B])}{\sqrt{\sum_{i \in I_A \cap I_B} (r_{A,i} - \text{Mean}[r_A])^2} \sqrt{\sum_{i \in I_A \cap I_B} (r_{B,i} - \text{Mean}[r_B])^2}}$$

i表示项Item，两个用户只有在有公共的评分项时，才有交集。
$\text{Mean}[r_A]$表示一个用户的平均分，$r_{A,i}$表示的是用户A的第i项的评分。

```
s[A_, B_] := Σ_{i∈I_A∩I_B} (r_{A,i} - Mean[r_A]) (r_{B,i} - Mean[r_B]) / (√(Σ_{i∈I_A∩I_B} (r_{A,i} - Mean[r_A])^2) √(Σ_{i∈I_A∩I_B} (r_{B,i} - Mean[r_B])^2))
```

```
asso[User]
```

```
⟨|User-A → A, User-B → B, User-C → C, User-D → D, User-E → E, User-F → F, User-G → G,
 User-H → H, User-I → I, User-J → J, User-K → K, User-L → L, User-M → M, User-N → N,
 User-O → O, User-P → P, User-Q → Q, User-R → R, User-S → S, User-T → T,
 User-U → U, User-V → V, User-W → W, User-X → X, User-Y → Y, User-Z → Z|⟩
```

```
assoFull[User] = AssociationThread[
   (StringTrim[#, "User-"] & /@ nameList[User]) → assoValue[User]];
```

建立KeyValue的关联结构后，就可以从用户名和宝贝ID进行取值

公式的验证上，因为不同的环境下，可能略有差异，比如是否标准化，标签化的范围，是否是[-1,1]等，因此可以输入参数资料中的数据，进行检验。

《推荐系统》Page9中的表2-1

```
userList = {Alice, 用户1, 用户2, 用户3, 用户4};
itemList = {Item1, Item2, Item3, Item4, Item5};

       5  3  4  4  No
       3  1  2  3   3
data = 4  3  4  3   5 ;
       3  3  1  5   4
       1  5  5  2   1
```

```
assoMatrix = Association[Thread[itemList ~ Rule ~ #]] & /@
  Association[Thread[userList ~ Rule ~ data]]
```

```
⟨|Alice → ⟨|Item1 → 5, Item2 → 3, Item3 → 4, Item4 → 4, Item5 → No|⟩,
 用户1→ ⟨|Item1 → 3, Item2 → 1, Item3 → 2, Item4 → 3, Item5 → 3|⟩,
 用户2→ ⟨|Item1 → 4, Item2 → 3, Item3 → 4, Item4 → 3, Item5 → 5|⟩,
 用户3→ ⟨|Item1 → 3, Item2 → 3, Item3 → 1, Item4 → 5, Item5 → 4|⟩,
 用户4→ ⟨|Item1 → 1, Item2 → 5, Item3 → 5, Item4 → 2, Item5 → 1|⟩|⟩
```

```
mean[list_] := Mean@Select[list, NumberQ]
```

```
filter[list_] := Select[list, NumberQ]
```

计算过程中要每个用户的平均分

```
meanList = N@Map[mean[Values[#]] &, assoMatrix, {1}]
```

```
⟨|Alice → 4., 用户1→ 2.4, 用户2→ 3.8, 用户3→ 3.2, 用户4→ 2.8|⟩
```

提取一个用户的某个物品的评分

```
assoMatrix[Alice, Item1]
```

```
5
```

实现公式中的分子部分，因为Listable属性，向量化操作等，许多时候不用去写一些循环来实现两个列表里的元素的两两作用等。

```
r1 = filter[Values[(assoMatrix[Alice] - meanList[Alice])]]
```

```
{1., -1., 0., 0.}
```

```
r2 = Values[(assoMatrix[用户1] - meanList[用户1])]
```

```
{0.6, -1.4, -0.4, 0.6, 0.6}
```

```
numerator = Total[filter[Values[(assoMatrix[Alice] - meanList[Alice])]
     Values[(assoMatrix[用户1] - meanList[用户1])]]]
```

```
2
```

```
filter[Values[(assoMatrix[Alice] - meanList[Alice])]
  Values[(assoMatrix[用户1] - meanList[用户1])]]
```

```
{0.6, 1.4, 0., 0.}
```

实现公式中的分母部分

```
assoMatrix[Alice] - meanList[Alice]
```

```
⟨|Item1 → 1., Item2 → -1., Item3 → 0., Item4 → 0., Item5 → -4. + No|⟩
```

```
√Total[(Values[filter@(assoMatrix[Alice] - meanList[Alice])])^2]
 √Total[(Values[filter@(assoMatrix[用户1] - meanList[用户1])])^2]
```

```
2.52982
```

```
√((5 - 4)^2 + (3 - 4)^2) √((3 - 2.4)^2 + (1 - 2.4)^2 + (2 - 2.4)^2 + (3 - 2.4)^2 + (3 - 2.4)^2)
```

```
2.52982
```

```
√((5 - 4)^2 + (3 - 4)^2) √((3 - 2.4)^2 + (1 - 2.4)^2 + (2 - 2.4)^2 + (3 - 2.4)^2)
```

```
2.38328
```

注意到，我们的物品集，当物品5排除后，则计算所有用户的时候均要排除特品5

因为filter之前的版本只是排除了Alice用户里的，因此是有问题的，

```
Total[(filter@Values[(assoMatrix[用户1] - meanList[用户1])])^2]
```

```
3.2
```

```
(assoMatrix[用户1] - meanList[用户1])^2
```

```
⟨|Item1 → 0.36, Item2 → 1.96, Item3 → 0.16, Item4 → 0.36, Item5 → 0.36|⟩
```

```
Total[r1^2]
```

```
2.
```

```
Total[r2^2 // Most]
```

```
2.84
```

```
√(2.84 × 2) // N
```

```
2.38328
```

```
2 / 2.38 // N
```

```
0.840336
```

数值为0.839

```
denominator = √Total[filter@Values[(assoMatrix[Alice] - meanList[Alice])^2]]
   √Total[Values[(assoMatrix[用户1] - meanList[用户1])^2]]
```

```
2.52982
```

```
numerator / denominator // N
```

```
0.790569
```

```
PearsonCorrelationTest[
 Values@assoFull[User][["A"]], Values@assoFull[User][["B"]]]
```

PearsonCorrelationTest::nortst :

At least one of the p−values in {0.0154548, 0.0154548}, resulting from a test for normality, is below 0.025`. The tests in {PearsonCorrelation} require that the data is normally distributed. ≫

```
0.579584
```

计算用户与所有宝贝的评分[查找用户的偏好宝贝]

```
assoFull[User]["A", 1]
```

```
0
```

```
assoFull[User]["A"]
```

```
⟨|1 → 0, 2 → 1, 3 → 1, 4 → 0, 5 → 2, 6 → 2, 7 → 3, 8 → 0, 9 → 0, 10 → 1|⟩
```

```
nameList[User]
```

```
{User-A, User-B, User-C, User-D, User-E, User-F, User-G, User-H,
 User-I, User-J, User-K, User-L, User-M, User-N, User-O, User-P, User-Q,
 User-R, User-S, User-T, User-U, User-V, User-W, User-X, User-Y, User-Z}
```