This demo application uses command line interface as the way of displaying movies recommended to the specified user. The app recommends the movies seen by the users who have similar behaviors with the user. To get similar users of the user, I use user-item based algorithm to implement Pearson Correlation Coefficient which is widely used in recommendation systems (for more details, please refer to <http://blog.csdn.net/wenbingoon/article/details/17414063>). In reality, to get the most precise results, tons of algorithms should be combined together. I just choose user based Pearson algorithm for simplicity and it’s believed a good way in this domain (like movie and music recommendation system).

There are 3 typical layers from architecture perspective. The client receives the user id and the number of movies to recommend. The recommendation includes movie title and IMDB and TMDB links for users to get details. The service layer is invoked by client and does business logics. The DAO layer communicate with the database. The application is managed by Spring 4 and the ORM implementation is hibernate 4. The database I use is Derby which is quite handy and integrated in IDE. I imported the small dataset to the database with some tables created.

The most time-consuming part in this app is user similarity computing because of massive records of user rating and tag history. I develop a client timer (by ScheduledExecutorService) to do the computation and update the results to a database table. This job runs at daily basis and it’s supposed to execute at midnight. Ideally, its’ better to use a server timer like (EJB timer) to do the job. Due to limited time, I just use the client timer. This user similarity records are maintained in a database table, so when the client request the recommendation for a user, the recommendation system can do the recommender algorism based on these similarity records instead of having to do the computation at that moment, which will save lots of waiting time for the user. As for the timer job, for every user, the computing last 10s around in my computer. So, for the small dataset, the job execution will last like 1 hour. For this demo, the time cost is acceptable. However, for the large dataset, performance tuning should be taken. Maybe a server cluster is a good choice.

Here is the code structure of the project:



The snapshot of running this demo with a valid user id 2:

Note: The list is different from the hello world app (https://github.com/lenskit/lenskit-hello) supplied by LensKit, which by default is using item-item based algorithm and get the movies which have the highest rating. And here I am using user-item based algorithm which I suppose makes more sense in this domain.



The data for similarity computing of user id 2:

