天津工业大学

毕业论文

协同过滤推荐算法及其模型的设计与开发

姓　　名　陈鹏

学　　院　理学院

专　　业　信息与计算科学

指导教师　王国庆

职　　称　副教授

2015 年 6 月 5 日

摘　要

协同过滤推荐，其原理是利用人们共同的喜好和行为为相似的人推荐他可能喜欢的信息，而用户的新的行为也会成为信息过滤的条件。1992 年，产生了最早应用协同过滤系统的设计 Tapestry，主要是解决 Xerox 公司在 Palo Alto 的研究中心资讯过载的问题。1994 年，产生了里程碑式的 GroupLens，这个系统主要是应用在新闻的筛选上，帮助新闻的阅听者过滤其感兴趣的新闻内容，阅听者看过内容后给一个评比的分数，系统会将分数记录起来以备未来参考之用。近年来，随着电子商务的兴起，电子商务的推荐系统也逐渐成熟，最早的成熟代表应属亚马逊网络书店，这就是著名的“买了这本书的人也买了什么”的算法。现在是一个信息严重过载的时代，从消息流中找寻自己的爱好的信息也比较困难。协同过滤推荐过滤掉不喜欢的资讯，推荐用户喜欢的资讯，节省时间提高效率。

本论文通过对推荐系统的研究,进一步的说明推荐系统在信息过载时代的重要性以及处理信息模型的精巧之处。本论文主要研究多维空间向量相似度的在推荐系统的应用。根据这个算法，结合 Java 的一些特性，设计一个类库，更方便的使用该算法进行过滤推荐。

关键词：协同过滤;Java;Web;反射;数据格式化;SpringMVC;Spring; Hibernate

ABSTRACT

The principle of collaborative filtering uses common preferences and behavior of people to recommend that he might prefer to similar information, and the new user behavior will become information filtering condition. In 1992,Tapestry as the first application of collaborative filtering system design was produced to resolve the Xerox research center mainly in Palo Alto information overload problem. In 1994,GroupLens was produced as a landmark.This system is mainly used in screening news to help its`s readers to find their interest news content.These readers will give a mark on the news they have read.The system will also refer to these marks for future reference.And in recent years, with the rise of e-commerce, e-commerce recommendation system gradually mature, and the symbol of mature earliest representatives should belong to the Amazon online bookstore. This is the famous algorithm — "people who bought this book also bought what". Now is an era of information overload time,and it`s more difficult to find their own interesting message from the message stream.Collaborative filtering can filter out the information that the users dislike,recommend users information they like,and can also save users` time,improve the efficiency.

In this paper, I explain the importance of recommendation system in the era of information overload and the exactitude of the information model handler with the study of recommendation system. The mainly research in this paper is on the similarity of two multidimensional space vectors in recommended system.According to this algorithm,combined with some of the features of Java, I will design and write a Java library to use Collaborative Filtering Recommendation conveniently.

Keywords：Collaborative Filtering Recommendation; Java; Web; Reflect; Data Format; SpringMVC; Spring; Hibernate

目 录

第一章 绪论 1
1.1 本设计的背景 1
1.2 本设计的目的和意义 1
1.3 本设计的主要内容 1
第二章 协同过滤算法的原理 3
2.1 协同过滤算法简介 3
2.2 收集用户信息 4
2.3 最近邻搜索（找到相似的用户） 4
2.4 计算推荐，产生推荐结果 6
第三章 余弦相似度之 Java 类库设计 7
3.1 余弦相似度之 Java 类库用到的技术 7
3.2 协同过滤算法之 Java 类库设计方案 11
3.3 协同过滤算法之 Java 类库用法 13
第四章 相关 Web 模块设计 15
4.1 MVC 模式 15
4.2 用到的其它组件 16
4.3 业务逻辑 17
4.4 开发平台 19
第五章 测试数据以及测试方法 22
5.1 测试数据来源 22
5.2 测试过程、方法及结果 22
第六章 结束语总结 23
参考文献 24
附录 25
谢辞 29

第一章　绪论

1.1 本设计的背景

协同过滤推荐，其原理是利用人们共同的喜好和行为为相似的人推荐他可能喜欢的信息，而用户的新的行为也会成为信息过滤的条件。1992 年，产生了最早应用协同过滤系统的设计 Tapestry，主要是解决 Xerox 公司在 Palo Alto 的研究中心资讯过载的问题。1994 年，产生了里程碑式的 GroupLens，这个系统主要是应用在新闻的筛选上，帮助新闻的阅听者过滤其感兴趣的新闻内容，阅听者看过内容后给一个评比的分数，系统会将分数记录起来以备未来参考之用。近年来，随着电子商务的兴起，电子商务的推荐系统也逐渐成熟，最早的成熟代表应属亚马逊网络书店，这就是著名的“买了这本书的人也买了什么”的算法。现在是一个信息严重过载的时代，从消息流中找寻自己的爱好的信息也比较困难。协同过滤推荐过滤掉不喜欢的资讯，推荐用户喜欢的资讯，节省时间提高效率。

但是，直接用的话不是很方便，随时切换各种实现也不方便。为了弥补这些缺陷，使人们更好的在面对海量数据时使用协同过滤算法，故设计了该组件，让人们更好的更方便的使用协同过滤推荐算法。

1.2 本设计的目的和意义

现在是一个信息严重过载的时代，从消息流中找寻自己的爱好的信息也比较困难。协同过滤推荐过滤掉不喜欢的资讯，推荐用户喜欢的资讯，节省时间提高效率。

本文通过对推荐系统的研究，进一步的说明推荐系统在信息过载时代的重要性以及处理信息模型的精巧之处。本论文主要研究多维空间向量相似度的在推荐系统的应用。

1.3 本设计的主要内容

本系统包括一个协同过滤推荐组件，默认包括了算法实现，并自行设计了一个算法，弥补某算法的一些不足，预留有接口，可以进行扩充其他推荐的算法。另外，还包括一个基于 B/S(Browser/Server)的 Web 系统，使用部分电影及其电影观看者的数据，作为该组件的 Demo 演示。

该组件使用了 Java 的高级特性反射（JAVA Reflection）和注解。反射是 Java 的一个高级特性，在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性。虽然反射

会丧失一部分运行效率，但是带来了更快的开发效率，经常见于一些库和框架中。例如 Spring 的 IoC 容器就是用了反射这个特性。

在作为测试作用的 Web 系统中，使用了 SpringMVC，Spring 和 MyBatis 框架。数据库使用 MySQL，前端页面使用 Velocity。

为什么使用 Velocity 而不是 JSP(Java Server Pages)?

Velocity 是一种模版引擎，JSP 本身是 Java Servlet，在使用时要经过.jsp->.java->.class 等多个步骤，而 Velocity 则在速度上更快，因为它不需要像解释 JSP 页面那么多的步骤；JSP 的优势是功能强大支持 Java 代码编写嵌入，支持 JSP 标签和 EL 表达式语言，还有就是官方标准，有丰富的第三方 JSP 标签库。经过斟酌，本次使用 Velocity 前端模版。

为什么使用 MyBatis 作为本示例的 ORM 映射框架？

在该项目示例中主要用到查询，并且是复杂查询。使用原生 SQL 比较方便。而这正是 SQL Mapper 框架 MyBatis 所擅长的。Hibernate 简单的增删改查很方便，主要使用 HQL，复杂的多表关联查询不方便，容易出现性能问题。故使用 MyBatis。

数据来自著名的 MovieLens，本例子中采用了 MovieLens 的子集。MovieLens 是一个 推荐系统和虚拟社区网站，其主要功能是应用协同过滤推荐技术提供数据，数据形式是电影，利用用户对电影的喜好，向用户推荐电影。该网站是 GroupLens Research 实验室的一个项目,GroupLens Research 实验室隶属于明尼苏达大学大学计算机系，MovieLens 创建于 1997 年。MovieLens 保存有用户对电影的评分，由此可进行个性化的协同过滤（推荐）。

第二章　协同过滤算法的原理

2.1 协同过滤算法简介

协同过滤(Collaborative Filtering)，简单说来就是利用兴趣类似，习惯类似的用户群体的喜好来推荐用户感兴趣的信息，个人通过参与使用，例如浏览，收藏，关注，点赞，踩，打分，转发，屏蔽等，也会给别人带来推荐的参考，帮助别人筛选过滤信息，参与的记录不一定限定于感兴趣的内容，特别是不感兴趣的参与记录也很重要，都会形成过滤的参考。重要的使用首先是使用在电子商务领域，重要的“买了某商品的用户有 N%还购买了某些商品”的算法，就是亚马逊首先使用，并取得了很好的导购效果，帮助人们找到想买的东西，刺激了消费，提高了成交量，节省了人们查询的时间。不只是电子商务，搜索引擎、豆瓣/网易 FM，豆瓣电影，以及一些读书类社区，都应用了此算法，极大的提高了效率。

协同过滤的发展历史：

1992 年，产生了最早应用协同过滤系统的设计 Tapestry，主要是解决 Xerox 公司在 Palo Alto 的研究中心资讯过载的问题。1994 年，产生了里程碑式的 GroupLens，这个系统主要是应用在新闻的筛选上，帮助新闻的阅听者过滤其感兴趣的新闻内容，阅听者看过内容后给一个评比的分数，系统会将分数记录起来以备未来参考之用。近年来，随着电子商务的兴起，电子商务的推荐系统也逐渐成熟，最早的成熟代表应属亚马逊网络书店，这就是著名的“买了这本书的人也买了什么”的算法。

协同过滤分类：用户（User-based）、项目（Item-based）、模型（Model-based）三种基础的协同过滤。下面主要讨论以用户为基础的协同过滤。

以用户为基础的协同过滤，用相似统计的方法得到具有相似使用习惯、兴趣、喜好厌恶的相邻用户。方法步骤：

1) 收集用户习惯
   收集用户产生的喜好信息，可以是用户主动参与产生的数据，例如评分，点赞，收藏，踩等，也可以是被动产生的，例如购买，浏览等。
2) 最近邻搜索（找到相似的用户）
   以用户为基础的协同过滤的协同过滤的一个重要过程就是找到习惯喜好相似的用户，那就需要计算两个用户的相似度。例如 A 对某些电影的评分记录是{(a,5)(b,1)(c,3)(d,4)…}，B 对某些电影的评分记录是{(a,4)(c,4)(d,4)(e,2)…}，C 也有相应的评分记录。那么 A 和 B 的相似度量化，A 与 BC 哪一个更相似，这就是相似度算法需要解决的问题。目前使用较多的相似度算法有皮尔逊相关系数(Pearson Correlation Coefficient)，余弦相似度(Cosine-based Similarity)，调整余弦相似度算法(Adjusted

Cosine Similarity)等。

3) 计算推荐，产生推荐结果
有了最近邻集合，就可以对该集合的兴趣进行聚合排序，产生推荐结果。常见的策略有 Top-N 推荐。Top-N 推荐是针对个体用户产生，对每个人产生不一样的结果。

本论文主要实现了以用户为基础的的协同过滤，实现步骤按该过滤的三个步骤依次实现。

2.2 收集用户信息

收集用户产生的喜好信息，可以是用户主动参与产生的数据，例如评分，点赞，收藏，踩等，也可以是被动产生的，例如购买，浏览次数，用户页面停留时间等。例如，在淘宝收藏（主动）了某件商品，浏览（被动）了某件商品等，很快系统就会根据你的收藏或者浏览对你进行精准个性化推荐。

2.3 最近邻搜索（找到相似的用户）

以用户为基础的协同过滤的协同过滤的一个重要过程就是找到习惯喜好相似的用户，那就需要计算两个用户的相似度。例如 A 对某些电影的评分记录是{(a,5)(b,1)(c,3)(d,4)…}，B 对某些电影的评分记录是{(a,4)(c,4)(d,4)(e,2)…}，C 也有相应的评分记录。那么 A 和 B 的相似度量化，A 与 BC 哪一个更相似，这就是相似度算法需要解决的问题。目前使用较多的相似度算法有皮尔逊相关系数(Pearson Correlation Coefficient)，余弦相似度(Cosine-based Similarity)，调整余弦相似度算法(Adjusted Cosine Similarity)等。

下面讨论相似度计算。

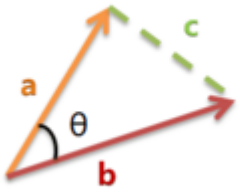
相似度的计算，大部分是基于多维空间向量(Vector)的，其实也就是两个向量的距离，距离越近，两个向量越相似。在一般的推荐场景中，在用户和物品偏好的二维矩阵中，我们可以将一个用对这些物品的喜好作为一个向量，其他用户同样可以量化成向量，来计算代表用户喜好的向量之间的相似度。同样的，将所有的用户对某个物品的喜好等作为一个向量来衡量物品之间的相似度。下面介绍几种常见的相似度计算方法。

1) 皮尔逊相关系数（Pearson Correlation Coefficient）
皮尔逊相关系数一般用于计算两个定距变量间联系的紧密程度，它的取值在 [-1，+1] 之间。其中，1 表示变量完全正相关， 0 表示无关，-1 表示完全负相关。计算公式如下：

$$p(x,y)=\frac{\sum x_i y_i - n\bar{x}\bar{y}}{(n-1)s_x s_y}=\frac{n\sum x_i y_i-\sum x_i \sum y_i}{\sqrt{n\sum x_i^2-(\sum x_i)^2}\sqrt{n\sum y_i^2-(\sum y_i)^2}}$$

2) 余弦相似度（Cosine-based Similarity）

余弦相似度又称为余弦相似性。通过计算两个向量的夹角余弦值来评估他们的相似度。具体过程是将向量根据坐标值，绘制到向量空间中。如最常见的二维空间。求得他们的夹角，并得出夹角对应的余弦值，此余弦值就可以用来表征，这两个向量的相似性。夹角越小，余弦值越接近于 1，它们的方向更加吻合，则越相似。



设向量 $A=(A_1,A_2,...,A_n)$，$B=(B_1,B_2,...,B_n)$ 。在多维空间内：

$$\cos\theta=\frac{\sum_{i=1}^{n}(A_i\times B_i)}{\sqrt{\sum_{i=1}^{n}(A_i)^2}\times\sqrt{\sum_{i=1}^{n}(B_i)^2}}=\frac{A^T\cdot B}{\|A\|\times\|B\|}$$

余弦值的范围在[-1,1]之间，值越趋近于 1，代表两个向量的方向越趋近于 0，他们的方向更加一致。相应的相似度也越高。

3) 欧几里德距离的相似度（Euclidean Distance-based Similarity）

假设 x，y 是 n 维空间的两个点，它们之间的欧几里德距离是：

$$d(x,y)=\sqrt{(\sum(x_i-y_i)^2)}$$

可以看出，当 n=2 时，欧几里德距离就是平面上两个点的距离。计算出来的欧几里德距离是一个大于 0 的数，为了使其更能体现用户之间的相似度，可以把它规约到(0, 1]之间，具体做法为：1 / (1 + d)。该数值表示距离越小，相似度越大。

$$sim(x,y)=\frac{1}{1+d(x,y)}$$

4) 取模余弦相似度(自行设计)

背景：

余弦相似度只能反映两个向量角度上的相似度，而长度上的相似度却没有表示，这在有些场合会不适合。例如，用户 A 对 m,n 两个电影的评分

为(10,5),用户 B 对 m,n 两个电影的评分为(2,1),那么 AB 两个用户进行余弦相似度分析的话相似度为 1，但是两个用户并不相似：用户 A 对电影 m 很喜欢，对电影 n 评价为一般；而用户 B 对这个两个电影都不喜欢，可见并不相似。故设计此算法。

优点：

兼顾角度和长度绝对值；

使用到的计算值均为之前的计算中间值，合理控制计算规模的增加。

原理：

$$MyCosSimilarity = \cos\theta * ModuloRate$$

其中，

- ModuloRate 为 对两个向量进行取模、除运算，如果大于 1 取其倒数；
- cosθ 为余弦相似度；
- MyCosSimilarity 的 取值范围为 [-1,1]，越接近 1 越相似。

优点：

- ✓ 兼顾角度和长度绝对值；
- ✓ 使用到的计算值均为之前的计算中间值，合理控制计算规模的增加。

2.4 计算推荐，产生推荐结果

有了最近邻集合，就可以对该集合的兴趣进行聚合排序，产生推荐结果。常见的策略有 Top-N 推荐。Top-N 推荐是针对个体用户产生，对每个人产生不一样的结果。

第三章　余弦相似度之 Java 类库设计

3.1 余弦相似度之 Java 类库用到的技术

用 Java 实现该类库，力求简单可依赖，容易拓展。

首先给出具体的包结构和相关类，如下：

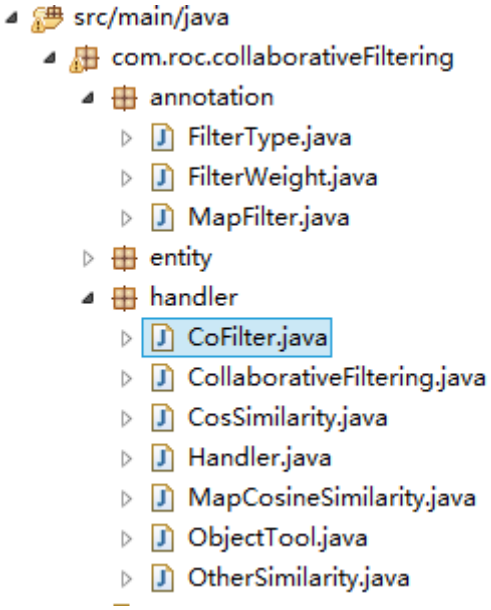


图 1-项目结构图

类、接口继承实现关系如下：

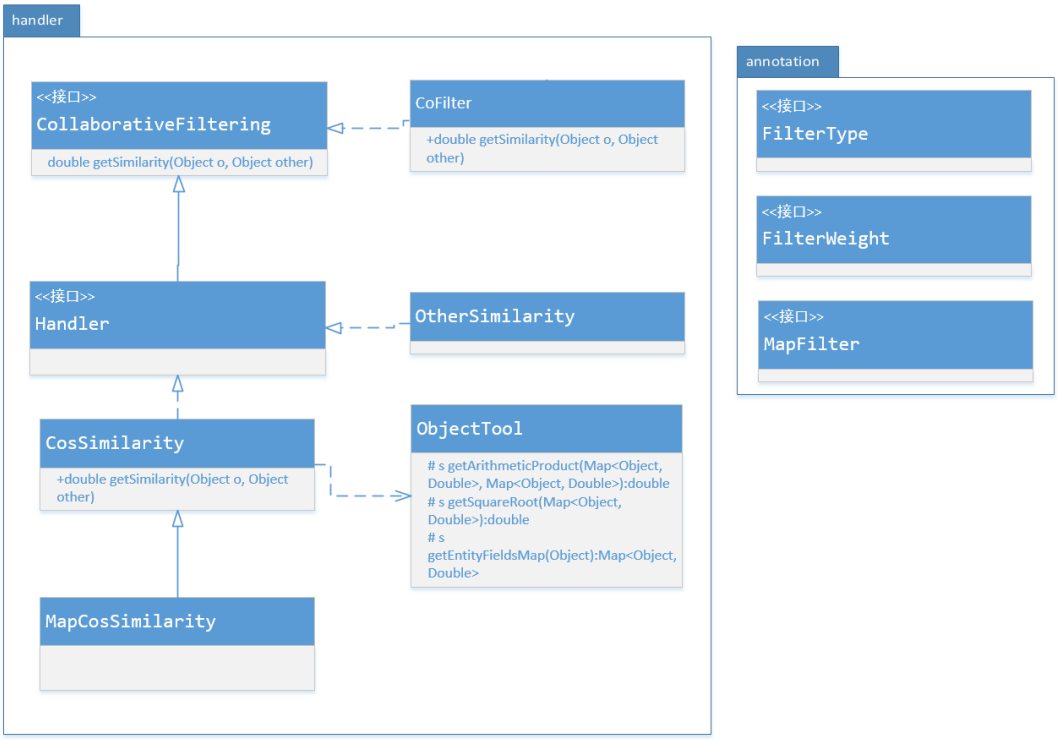


图 2-类和接口继承关系

本类库中不但用到了常见的面向对象特性，集成封装多态，还用到了很多的 Java 高级特性，如 Java 反射，Java 注解与自定义注解，动态代理等。

3.1.1 Java 反射机制

反射机制（Reflection）就是可以把一个类,类的成员(方法,属性),当成一个对象来操作,通过 Java 的反射 API 可以在运行时刻获取和修改其内部结构，还可以执行其方法。知道了 Java 类的内部结构之后，就可以与它进行交互，包括创建新的对象和调用对象中的方法等。这种交互方式与直接在源代码中使用的效果是相同的，但是又额外提供了运行时刻的灵活性。使用反射的一个最大的弊端是性能比较差。相同的操作，用反射 API 所需的时间大概比直接的使用要慢一两个数量级。不过现在的 JVM 实现中，反射操作的性能已经有了很大的提升。在灵活性与性能之间，总是需要进行权衡的。应用可以在适当的时机来使用反射 API。

Java 反射 API 的一个主要作用是获取运行时（Runtime）的对象信息。这对于程序的检查工具和调试器来说，是非常实用的功能。只需要短短的十几行代码，就可以遍历出来一个 Java 类、对象的内部结构，包括其中的构造方法、声明的域和定义的方法等。这不得不说是一个很强大的能力。只要有了 java.lang.Class 类 的对象，就可以通过其中的方法来获取到该类中的构造方法、域和方法。对应的方法分别是 getConstructor、getField 和 getMethod。这三个方法还有相应的

getDeclaredXXX 版本，区别在于 getDeclaredXXX 版本的方法只会获取该类自身所声明的元素，而不会考虑继承下来的。Constructor、Field 和 Method 这三个类分别表示类中的构造方法、域和方法。这些类中的方法可以获取到所对应结构的元数据。

3.1.2 Java 注解与自定义注解

自 J2SE5.0 引入注解后，注解成为了 Java 平台中非常重要的一部分。我们也经常在应用代码中经常用到@Override（显式覆盖重写，编译器检查使用）等，很多 Java 的类库框架也会大量的使用自定义注解，极大的方便了开发者。

元注解：

元注解负责注解其他注解。J2SE5.0 版本在 java.lang.annotation 提供了四种元注解，专门注解其他的注解：

@Documented：注解信息是否将包含在 JavaDoc 中
@Retention：定义该注解的生命周期
@Target：注解的作用范围
@Inherited：是否允许子类继承该注解

@Documented：一个简单的 Annotations 标记注解，表示是否将注解信息添加在 java 文档中。

@Retention：@Retention 定义了该 Annotation 被保留的时间长短：某些 Annotation 仅出现在源代码中，而被编译器丢弃；而另一些却被编译在 class 文件中；编译在 class 文件中的 Annotation 可能会被虚拟机忽略，而另一些在 class 被装载时将被读取（请注意并不影响 class 的执行，因为 Annotation 与 class 在使用上是被分离的）。使用这个 meta-Annotation 可以对 Annotation 的“生命周期”限制。

作用：表示需要在什么级别保存该注释信息，用于描述注解的生命周期（即：被描述的注解在什么范围内有效）

取值（RetentionPoicy）有：

1.SOURCE:在源文件中有效（即源文件保留）
2.CLASS:在 class 文件中有效（即 class 保留）
3.RUNTIME:在运行时有效（即运行时保留）

Retention meta-annotation 类型有唯一的 value 作为成员，它的取值来自 java.lang.annotation.RetentionPolicy 的枚举类型值。

@Target：表示该注解的作用范围。如果不明确指出，该注解可以放在任何地方。以下是一些可用的参数。需要说明的是：属性的注解是兼容的，如果你想给 7 个属性都添加注解，仅仅排除一个属性，那么你需要在定义 target 包含所有的属性。

ElementType.TYPE:用于描述类、接口或 enum 声明
ElementType.FIELD:用于描述实例变量
ElementType.METHOD:用于描述实例方法
ElementType.PARAMETER:用于描述参数
ElementType.LOCAL_VARIABLE:用于描述局部变量
ElementType.CONSTRUCTOR:用于描述构造器
ElementType.PACKAGE：用于记录 java 文件的 package 信息

在本类库中用到的元注解包括但不限于：@Retention(RetentionPolicy.RUNTIME)，@Target(ElementType.TYPE)，@Target(ElementType.FIELD)。

自定义注解：

使用@interface 自定义注解时，自动继承了 java.lang.annotation.Annotation 接口，由编译程序自动完成其他细节。在定义注解时，不能继承其他的注解或接口。@interface 用来声明一个注解，其中的每一个方法实际上是声明了一个配置参数。方法的名称就是参数的名称，返回值类型就是参数的类型（返回值类型只能是基本类型、Class、String、enum）。可以通过 default 来声明参数的默认值。

定义注解格式：
public @interface 注解名 {定义体}

注解参数的可支持数据类型：

1.所有基本数据类型（int,float,boolean,byte,double,char,long,short)
2.String 类型
3.Class 类型
4.enum 类型
5.Annotation 类型
6.以上所有类型的数组(一维数组)

Annotation 类型里面的参数该怎么设定：

第一,只能用 public 或默认(default)这两个访问权修饰.例如,String value();这里把方法设为 defaul 默认类型；

第二,参数成员只能用基本类型 byte,short,char,int,long,float,double,boolean 八种基本数据类型和 String,Enum,Class,annotations 等数据类型,以及这一些类型的数组.例如,String value();这里的参数成员就为 String；

第三,如果只有一个参数成员,最好把参数名称设为"value"，后加小括号。

3.1.3 面向对象三大特性之多态

什么是多态？

- 面向对象三大特性：封装、继承、多态。从一定角度来看，封装和继承几乎都是为多态而准备的。
- 多态的定义：指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同而采用多种不同的行为方式。（发送消息就是函数调用）
- 实现多态的技术称为：动态绑定（dynamic binding），是指在执行期间判断所引用对象的实际类型，根据其实际的类型调用其相应的方法。结合上边的反射和注解，可以动态的绑定所需要的类型。
- 多态的作用：消除类型之间的耦合关系。

多态存在的三个必要条件：

1) 继承（也可以是实现某接口，此时接口引用即是父引用）
2) 重写（覆盖）
3) 父引用指向子类对象

3.2 协同过滤算法之 Java 类库设计方案

主要的设计思想是**简单，可依赖，易拓展**。
本类库主要有两个模块：注解标记和计算相似度。

3.2.1 自定义注解接口

FilterType 接口

本注解标记在需要求相似度的类上，被标记的类生成的对象在运行时依然保持注解信息。代码如下：

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface FilterType {
    Class<? extends Handler> value() default CosSimilarity.class;
}
```

其中声明了一个方法，可以存放注解，一个继承（实现）Handler 接口的类，该

类负责处理两个需要计算相似度的对象，通过 getSimilarity()方法返回一个数值。有默认值，默认处理类是 CosSimilarity。

FilterWeight 接口
本注解标记在字段（Filed）上，运行时保持注解信息。代码如下：

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface FilterWeight {
   int value() default 1;
}
```

该注解表示某个字段（Filed）的权重，默认值是 1。

MapFilter 接口
本注解标记在字段（Filed）上，运行时保持注解信息。代码如下：

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface MapFilter {}
```

该注解表示某个字段（Filed）是存储多维向量的 Map，Map 的键值对分别是<向量维度名称，该维度长度>。

3.2.2 计算余弦相似度的实现

余弦相似度计算公式：

$$\cos\theta = \frac{\sum_{i=1}^{n}(A_i \times B_i)}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}} = \frac{A^T \cdot B}{\|A\| \times \|B\|}$$

图 3-余弦相似度计算公式

CosSimilarity 类实现如下：

```
public double getSimilarity(Object o, Object other) {
    Map<Object, Double> oMap = ObjectTool.getEntityFieldsMap(o);
    Map<Object, Double> otherMap = ObjectTool.getEntityFieldsMap(other);
    double upLine = ObjectTool.getArithmeticProduct(oMap, otherMap);
    double downLine = ObjectTool.getSquareRoot(oMap)
            * ObjectTool.getSquareRoot(otherMap);
    return upLine / downLine;
}
```

首先将每个参数Map化，键表示向量的维度名字，值表示该维度的长度，这样用两个Map表示了两个向量，也就是两个对象的特性描述。

然后计算两个向量的余弦相似度，按公式计算即可。如果有的向量维度对方没有，则按对方在该维度长度为0。

3.2.3 其他相似度计算的实现

由于该类库预留有接口，其他相似度计算方法可以通过扩展实现，实现Handler接口即可。

3.3 协同过滤算法之 Java 类库用法

1) 属性传递向量信息：

需要计算的对象类：

```
@FilterType(CosSimilarity.class)
public class EntityDemo {
    @FilterWeight(5)
    private int age;
    @FilterWeight
    private int sex;

    private int id;
    // setter and getter...
}
```

描述：
@FilterType为CosSimilarity.class，则使用CosSimilarity.class处理该类的实例对象。

age的注解信息为**@FilterWeight(5)**，表示age（年龄）权重为5。
sex的注解信息为**@FilterWeight**，表示sex（性别）使用权重是默认值，默认值是1。
id没有注解，不参与向量相似度计算。

```
public class App {
    public static void main(String[] args) {
        CollaborativeFiltering collaborativeFiltering = new CoFilter();
        EntityDemo o = new EntityDemo();
        o.setSex(1);
        o.setAge(23);
        EntityDemo o1 = new EntityDemo();
        o1.setSex(0);
        o1.setAge(17);
        double d = collaborativeFiltering.getSimilarity(o, o1);
        System.out.println("similar:" + d);
}
```

Console打印输出为：similar:0.9999621949605748。非常相似了。

2) 直接使用Map传递向量信息：

需要计算的对象类：

由于和上边类似，不再赘述。

```
@FilterType(MapCosineSimilarity.class)
public class MapEntity {
    private int id;
    @MapFilter
    // 前者为向量名字，后者是向量的长度
    private Map<Object, Double> map = new HashMap<Object, Double>();
    // setter and getter...

}
```

由于和上边类似，不再赘述。

第四章　相关 Web 模块设计

4.1 MVC 模式

MVC(Model View Controller)，是模型-视图-控制器的缩写，广泛应用于软件开发，是一种软件设计典范，很好的实现了业务逻辑，数据、界面显示分离的方法组织代码，修改一部分如界面时，一般不会影响业务逻辑，不需要重新编写业务逻辑。

4.1.1 MVC 及其它组件

本次测试数据使用著名的MovieLens数据，该数据为电影数据，里面有用户的评分记录，十分适合于协同过滤测试。

数据下载页面：http://grouplens.org/datasets/movielens/

4.1.2 Model 说明

因为数据是基于电影的，所以 Model 实体类均是电影相关，当然还有用户类。

Movie	电影
MovieRating	用户对电影评分
User	用户

4.1.3 View 说明

View 使用 Web 页面，使用 Velocity 模版引擎生成，含有 Velocity 标签。使用了 Bootstrap3.3.4 前段框架，搭配 Javascript+CSS+DIV，开发迅速，效果简洁美观。

4.1.4 Controller 说明

Controller 使用了 SpringMVC 框架，天生与 Spring 集成。优点包括：

- 轻松写出设计出干净的 Web 层和薄薄的 Web 层

- 支持灵活的 URL 到页面控制器的映射
- 提供强大的约定大于配置的契约式编程支持
- 能简单的进行Web层的单元测试
- 非常容易与其他视图技术集成，如Velocity、FreeMarker等
- 支持Restful风格

示例：

```
@RequestMapping(value = "/", method = RequestMethod.GET)
public String home(Model model) {
    model.addAttribute("topUsers", dbUtil.getToperUsers());
    return "index";
}
```

“/”表示根域名；method = RequestMethod.GET表示请求方法为Get；设置模型数据为topUsers，表示打分给电影数量最多的用户；返回字符串“index”，表示返回视图index，视图的位置在Spring配置文件里。

```
<beans:bean id="velocityConfig"
    class="org.springframework.web.servlet.view.velocity.VelocityConfigurer">
    <beans:property name="configLocation" value="classpath:velocity.properties" />
    <beans:property name="resourceLoaderPath" value="/WEB-INF/templates" />
</beans:bean>
<beans:bean id="viewResolver"
    class="org.springframework.web.servlet.view.velocity.VelocityViewResolver">
    <beans:property name="suffix" value=".vm" />
    <beans:property name="contentType" value="text/html;charset=UTF-8" />
    <beans:property name="requestContextAttribute" value="rc" />
</beans:bean>
```

视图模版在“/WEB-INF/templates”下，后缀为“.vm”。即该视图文件为：“/WEB-INF/templates/index.vm”。

4.2 用到的其它组件

Junit：一个 Java 语言的单元测试框架。可以用它对写好的每一个功能进行方便的单元测试，DEBUG，同时可以将测试代码和产品代码分开，结合 Maven 可以在进行打包发布的时候防止测试代码被编译打包发布。

MyBatis：MyBatis 是支持定制化 SQL、存储过程以及高级映射的持久层框架。MyBatis 避免了几乎所有的 JDBC 代码和手工设置参数以及抽取结果集。MyBatis 使用简单的 XML 或注解来配置和映射基本体，将接口和 Java 的

POJOs(Plain Old Java Objects,普通的 Java 对象)映射成数据库中的记录。本项目中主要使用了注解形式，使用 Native SQL（数据库是 MySQL）非常简单快捷。举例如下：

```
public interface CoFilterRepository {
    @Select("SELECT * FROM movierating WHERE userid = #{userid}")
    List<MovieRating> getMovieRatingByUserId(@Param("userid") int userid);
    //其他的一些带有注解的接口...
}
```

这样就可以得到 List<MovieRating>了。

4.3 业务逻辑

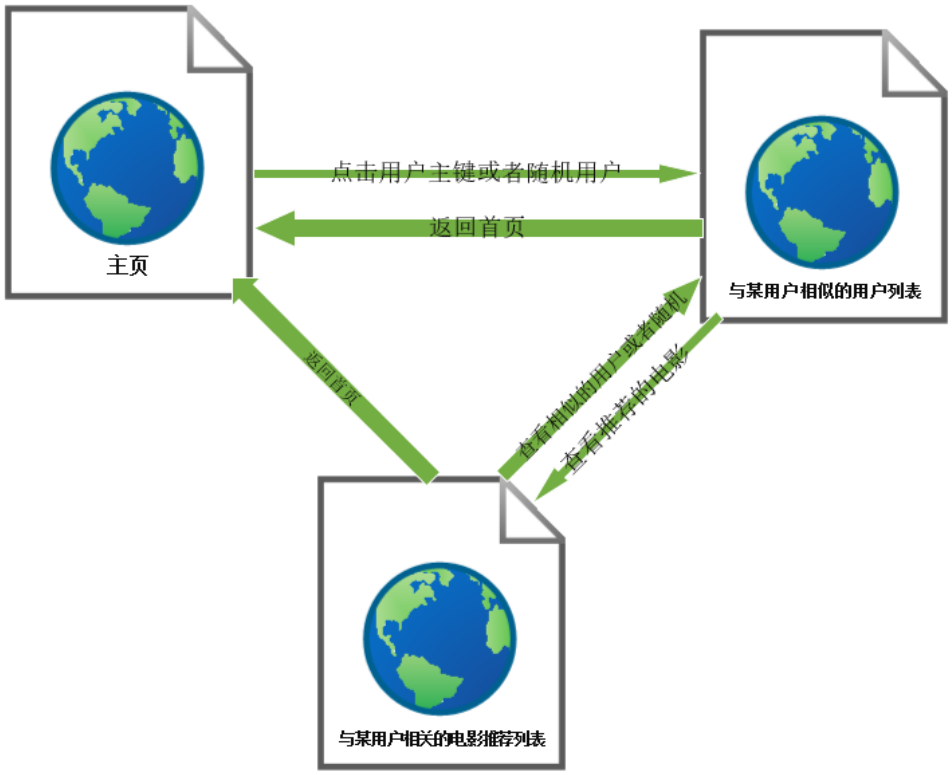


图 4-业务逻辑

图 5-首页

图 6-与某用户相似的用户列表

图 7-电影推荐列表

主要有以下几个功能：

- 查询打分给电影数量最多的用户列表，按其评过的电影数量降序排列。
- 点击用户主键查询得到相似用户列表。
- 点击随机会随机到数据库内的一个主键，得到该用户的相似用户列表；
- 点击推荐的电影会得到该用户的个性化推荐电影列表。

4.4 开发平台

4.4.1 IDE：Spring Tool Suite™ (STS)

STS 是 Spring 开源组织开发的 Spring 集成开发工具，基于 Eclipse，是一款开源免费集成开发工具。利用 STS，可以很简单的生成 Spring Project，例如 Web Project，而且工程里边的配置文件会自动生成，大大提高搭建环境的效率。目前 Windows 平台的最新版本是 3.6.4。

4.4.2 构建工具：Apache Maven

Apache Maven，是软件（特别是 Java 软件）的项目管理及自动构建工具，由 Apache 软件基金会所提供。POM 代表项目对象模型，它是 Maven 的基本单位，保存着各种配置。

STS 原生支持创建基于 Maven 的 Spring 项目，二者配合非常好。

协同过滤 Java 类库的 pom.xml 内容摘了如下：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.roc</groupId>
    <artifactId>collaborativeFiltering</artifactId>
    <version>0.0.1</version>
    <packaging>jar</packaging>

    <name>collaborativeFiltering</name>
    <url>http://maven.apache.org</url>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
    </dependencies>
</project>
```

此时使用STS的导出jar功能就可以导出一个只依赖jdk的jar包。

注意这三行：

```
    <groupId>com.roc</groupId>
    <artifactId>collaborativeFiltering</artifactId>
    <version>0.0.1</version>
```

使用这三行，就可以在别的项目中通过dependency添加对该协同过滤Java类库的依赖了，如下：
示例Demo的Web项目中的pom.xml

```
<dependency>
    <groupId>com.roc</groupId>
    <artifactId>collaborativeFiltering</artifactId>
    <version>0.0.1</version>
</dependency>
```

这样，这个Web项目就可以使用该协同过滤Java类库了。

第五章　测试数据以及测试方法

5.1 测试数据来源

数据来自著名的MovieLens，本例子中采用了MovieLens的子集。MovieLens是一个 推荐系统和虚拟社区网站，其主要功能是应用协同过滤推荐技术提供数据，数据形式是电影，利用用户对电影的喜好，向用户推荐电影。该网站是GroupLens Research实验室的一个项目,GroupLens Research实验室隶属于明尼苏达大学大学计算机系，MovieLens创建于1997年。MovieLens保存有用户对电影的评分，由此可进行个性化的协同过滤（推荐）。

5.2 测试过程、方法及结果

首先将数据格式化批量入库，数据库是 MySQL。然后编写相应的 SQL 语句查询到必要的信息，结合 MyBatis 和本文设计的协同过滤类库，得到相似的用户，并按照相似度排序。经验证，这些相似用户的确在评价习惯上也就是电影的喜好上比较相似。最后得出推荐结果。

大约 638 个用户、共 978 条评价记录多表关联可在建立索引后 100ms 之内过滤完毕，性能较好。

第六章　结束语总结

本文通过对协同过滤推荐算法、Java 高级特性以及 SpringMVC，Spring，MyBatis 等相关知识技术的深入研究，实现了一个基于 Java 的协同过滤推荐类库，并且给出了基于 SSI 架构的 Web 项目 Demo。方便大家使用、学习和研究。通过本文的深入研究，得出一下结论：

(1) 在设计类库时应充分考虑到使用的场景，大胆的设计，可以考虑使用高级特性，但使用高级特性不是目的，目的是方便更方便的使用该类库。
(2) 在开发的时候要充分考虑可扩展性，力求简单拓展，无论是类库还是其他。
(3) 采用 SpringMVC 可以十分轻松清晰的设计视图和具体逻辑的结构，方便前端和后端服务的对接，有利于后期系统拓展和维护。
(4) 采用 MyBatis 可以简单快捷的使用 Native SQL，优化数据库性能时比较方便。
(5) 使用 MVC 模式，可以使项目结构清晰，有利于功能的解耦，简化开发流程，专注开发具体的模块功能。

虽然开发出来的类库和该 Web 项目 Demo 有较多的优势，但是也存在一些问题。随着新的技术知识的不断涌现，要使程序能够高效运行，就要做进一步的研究和试验：

(1) 计算相似度涉及多维空间向量，需要的数据量比较大，尤其是在大数据时代，这就需要优化 MySQL 数据的存储结构，适当数据冗余，建立合适的索引，优化 IO 性能。还可以尝试使用 NoSQL 数据库，例如 PostgreSQL、MongoDB 等，提高性能。
(2) 计算相似度同样计算量也非常大，可以设计多线程分布式服务来解决这个问题，同时关注和研究新的高效的相似度计算算法，拓展在本类库中。

参考文献

[1] Baron Schwartz, Peter Zaitsev, Vadim Tkachenko 著，宁海元，周振兴，彭立勋 等 译. 高性能 MySQL(第 3 版). 电子工业出版社, 2013.4

[2] 戴克(Paul Deck) [美] 著，林仪明 崔毅译. Spring MVC 学习指南. 人民邮电出版社.2015.5.6

[3] Cay S.Horstmann[美], Gary Cornell[美]. Core Java 2. Prentice Hall PTR ,2006.5

[4] 上野 宣[日] 著，于均良 译. 图解 HTTP. 人民邮电出版社, 2014.05.01

[5] 张为. 基于 Struts 框架的 JavaWeb 应用研究[J]. 长沙电力学院学报. 2006.2

[6] Joshua Bloch(美) 著，杨春花，俞黎敏 译. Effective Java Second Edition[M]. 北京:机械工业出版社. 2009.1

[7] 项亮 著. 推荐系统实践. 人民邮电出版社,2012.6

附录

附件一：中英文翻译对照

Abstract Class Versus Interface in the JDK 8 Era

Dustin Marx

In The new Java 8 Date and Time API: An interview with Stephen Colebourne, Stephen Colebourne tells Hartmut Schlosser, “I think the most important language change isn’t lambdas, but static and default methods on interfaces.” Colebourne adds, “The addition of default methods removes many of the reasons to use abstract classes.” As I read this, I realized that Colebourne is correct and that many situations in which I currently use abstract classes could be replaced with interfaces with JDK 8 default methods. This is pretty significant in the Java world as the difference between abstract classes and interfaces has been one of the issues that vex new Java developers trying to understand the difference. In many ways, differentiating between the two is even more difficult in JDK 8.

There are numerous examples of online forums and blogs discussing the differences between interfaces and abstract classes in Java. These include, but are not limited to, JavaWorld‘s Abstract classes vs. interfaces, StackOverflow‘s When do I have to use interfaces instead of abstract classes?, Difference Between Interface and Abstract Class, 10 Abstract Class and Interface Interview Questions Answers in Java, As useful and informative as these once were, many of them are now outdated and may be part of even more confusion for those new to Java who start their Java experience with JDK 8.

As I was thinking about the remaining differences between Java interfaces and abstract classes in a JDK 8 world, I decided to see what the Java Tutorial had to say on this. The tutorial has been updated to reflect JDK 8 and the Abstract Methods and Classes has a section called “Abstract Classes Compared to Interfaces” that has been updated to incorporate JDK 8. This section points out the similarities and differences of JDK 8 interfaces with abstract classes. The differences it highlights are the accessibility of data members and methods: abstract classes allow non-static and non-final fields and allow methods to be public, private, or protected while interfaces’ fields are inherently public, static, and final, and all interface methods are inherently public.

The Java Tutorial goes on to list bullets for when an abstract class should be considered and for when an interface should be considered. Unsurprisingly, these are derived from the previously mentioned differences and have primarily to do with whether you need fields and methods to be private, protected, non-static, or not final (favor abstract class) or whether you need the ability to focus on typing without regard to implementation (favor interface).

Because Java allows a class to implement multiple interfaces but extend only one class, the interface might be considered advantageous when a particular implementation needs to be associated with multiple types. Thanks to the JDK 8's default methods, these interfaces can even provide default behavior for implementations.

A natural question might be, "How does Java handle a class that implements two interfaces, both of which describe a default method with the same signature?" The answer is that this is a compilation error. This is shown in the next screen snapshot which shows NetBeans 8 reporting the error when my class implemented two interfaces that each defined a default method with the same signature [String speak()].

netBeans8CompilerErrorMultipleInterfacesSameDefaultMethodSignatures

As the screen snapshot above indicates, a compiler error is shown that states, "class … inherits unrelated defaults for … from types … and …" (where the class name, defaults method name, and two interface names are whatever are specified in the message). Peter Verhas has written a detailed post ("Java 8 default methods: what can and can not do?") looking at some corner cases (gotchas) related to multiply implemented interfaces with default method names with the same signature.

Conclusion

JDK 8 brings arguably the abstract class's greatest advantage over the interface to the interface. The implication of this is that a large number of abstract classes used today can likely be replaced by interfaces with default methods and a large number of future constructs that would have been abstract classes will now instead be interfaces with default methods.

翻译文稿

JDK1.8 中 接口和抽象类的异同

Dustin Marx

在最新版本 Java1.8 中的变化里， Stephen Colebourne 告诉 Hartmut Schlosser，“我认为最重要的语言上的改变不是 Lambdas 表达式，而是在接口上的静态和默认方法”。ColeBoune 补充道，“默认方法功能的增加减少了许多使用抽象类的原因”。当我读到这里时，我意识到 Colebourne 是正确的，而且我通常用抽象类的许多场景可以通过 JDK1.8 的默认方法（default method）替换为 JDK1.8 接口。

在网上论坛和博客中讨论 Java 接口和抽象类异同的例子不胜枚举。这些讨论包括但不限于， JavaWorld ’s Abstract classes vs. interfaces ， StackOverflow 's When do I have to use interfaces instead of abstract classes? ， Difference Between Interface and Abstract Class ， 10 Abstract Class and Interface Interview Questions Answers in Java ，作为曾经有用的信息，许多现在已经过时了，可能会成为让 Java 体验更加混乱的部分。

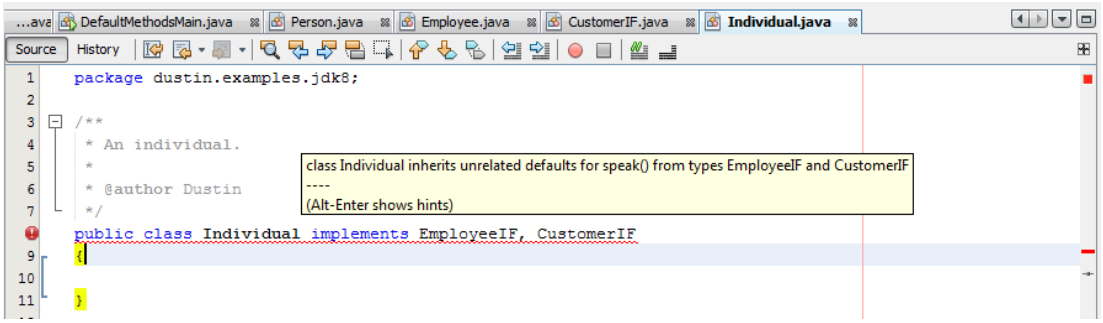
在我思考 Java1.8 中接口和抽象类尚存在的不同，我决定去看看 Java Tutorial 关于这部分说过的信息。这个引导（Tutorial）已经更新到 JDK 8 而且关于抽象方法和类的讨论有一个章节，称作 抽象类 vs 接口。这个章节指出在 JDK 8 中接口和抽象类的异同点。着重强调的不同点是数据成员和方法的作用域（访问权限）：

- 抽象类允许非静态和非 final 的域，允许方法成为 public,static 和 final
- 所有的接口方法本质为 public 的。

在 Java Tutorial 接着列出了许多应该考虑抽象类的场景，和许多应该考虑接口的场景。不出意料，这些源于前面提到的不同、分歧，主要处理你是否**需要域和方法成为 private, protected, non-static, 或者 not-final（偏向抽象类）**，或者你需要**把重点放在类型而不是考虑实现的能力（偏向接口）**。

因为 Java 语序一个类实现多个接口但只能继承一个类，接口可能被视为有利于多继承的实现。多亏 JDK 8 的默认方法，这些接口甚至能提供默认的实现。

那么，一个很自然的问题是：当一个类实现了两个接口，而这两个接口描述了一个具有相同特点的默认方法，Java 怎么处理。答案是，这会报一个编译错误。这是一个屏幕截图，NetBeans8 上运行报的这个错误。



编译报错

结论

JDK 8 争议性的把抽象类的优点带给接口。造成的影响是今天有大量的抽象类通过默认方法可能被替换为接口。

谢　辞

衷心的感谢我的导师王海庆老师。本文的研究工作是在王老师的悉心指导下完成的。感谢帮助过我的同学和朋友。还有我的每一位任课老师，谢谢你们！

四年时光转瞬即逝，大学的历程也快到终点了。这四年时间，不但使我的知识网络和科研能力上了一个台阶，更重要的是，培养了多方面的素质，培养了良好的思维习惯。

最后也是永远，感谢我的父母！每当我遇到困难的时候，父母总是第一个给我鼓励。每当遇到选择的时候，父母给出他们的见解，让我自己选择，并给我支持。回顾这些年在外求学的经历，总是有父母的激励在身边。他们在精神上和物质上的无私支持，坚定了我追求人生理想的信念。我相信我会达到自己的目标。

最后的大学作业，今天我满怀热情与感激地完成大学的最后一次作业，为我的大学生涯画上一个圆满的句号。毕业既是终点又是新的起点，整理好思绪和坚持，牢记老师的教诲、父母的嘱托，带着热情与激情继续出发。