# 摘要

# Abstract

# 第1章绪论

## 1.1研究背景和意义

### 1.1.1研究背景

### 1.1.2研究意义

## 1.2国内外发展状况

### 1.2.1国外研究现状

### 1.2.2国内研究现状

## 1.3论文内容结构

## 1.4本章小结

# 第2章个性化视频推荐系统相关技术

## 2.1 **系统开发环境及技术**

个性化视频推荐系统是用面向对象的解释性python语言开发的，在强大的pycharm IDE软件的支持下，利用开源的Django Web框架所开发，这样能够降低设计的成本，节省开发时间。

### 2.1.1 pycharm

PyCharm是由JetBrains开发的一款专业的Python IDE，可以帮助用户在使用Python语言开发时提高效率的工具，有调试，语法高亮，project管理，智能提示，单元测试和版本控制等功能，该IDE还支持一些高级功能，例如支持Django框架下的专业web开发。

### 2.1.2 django

Django是一个开放源代码的Web应用框架，由Python写。采用了MTV的框架模式，即模型M，模板T和视图V。



### 2.1.3 javascript

JavaScript是由网景公司与sun公司与1995年共同开发设计的一种直译式脚本语言，一种动态类型、弱类型、基于原型的语言。为了看上去能够像Java，因此取名为JavaScript。它包括ECMAScript：描述了该语言的语法和基本对象、文本对象模型（DOM）：描述处理网页内容的方法和接口和浏览器对象模型（BOM）：描述与浏览器进行交互的方法和接口三部分。

## 2.2 推荐系统相关技术

### 2.2.1 推荐系统的一般架构

推荐系统一般分为三个模块，推荐模块，UI模块，日志模块。推荐模块主要为推荐引擎，推荐引擎的构建可以来源于多个数据源（用户特征多样化）+不同的推荐算法，推荐引擎的架构可以是多样化的（实时推荐+离线推荐），最后融合推荐结果，融合方式可以是线性加权或者切换式的。UI模块一般包含三块：1）通过一定的方式推荐物品（物品标题、缩略图、简介）；2）给出推荐理由；3）数据反馈改进个性化推荐模型。日志模块主要采集用户反馈的结果，为推荐模块提供有效的数据，方便训练符合用户特点的模型给用户推荐恰当的物品。



## 2.3 推荐算法分析

有推荐功能的系统相比普通系统当然是多了推荐功能，推荐系统的核心是推荐算法，常见的推荐算法有基于内容、基于协同、关联规则、基于效用、基于知识和组合推荐等等。接下来对常见的算法做一描述。



### 2.3.1 基于内容

基于内容的基本思想是，根据用户过去喜欢的物品，为用户推荐和他过去喜欢的物品相似的物品。例如：在京东当中，每当进入任何一个物品页面的时候都会有一个“为你推荐”栏目，这时候他就会根据你经常购买的物品给你推荐相似的物品。对于我经常购买互联网类的书籍，所以他会给我推荐类似的书籍。

### 2.3.2 基于协同

协同过滤是比较著名的推荐算法。算法通过对用户历史行为数据的挖掘发现用户的偏好，基于不同的偏好对用户进行群组划分并推荐品味相似的商品。协同过滤推荐算法分为两类，分别是基于用户的协同过滤算法，和基于物品的协同过滤算法。简单的说就是：人以类聚，物以群分。

### 2.3.3 关联规则

关联规则是反映一个事物与其他事物之间的相互依存性和关联性，常用于实体商店或在线电商，通过对顾客的购买记录数据库进行关联规则挖掘，最终目的是发现顾客群体的购买习惯的内在共性。最经典的应用案例莫过于“啤酒和尿布”。

### 2.3.4 基于效用

基于效用的推荐是建立在对用户使用项目的效用情况上计算的，其核心问题是怎么样为每一个用户去创建一个效用函数，因此，用户资料模型很大程度上是由系统所采用的效用函数决定的。

### 2.3.5 基于知识

基于知识的推荐在某种程度是可以看成是一种推理技术，它不是建立在用户需要和偏好基础上推荐的。基于知识的方法因它们所用的功能知识不同而有明显区别。

### 2.3.6 组合推荐

金无足赤，人无完人。当然，算法也不列外，以上各个算法都有优缺点（见2.3.7 推荐算法比较），在实际运用当中，可以把某几个算法相结合，通过组合避免或者弥补推荐算法的弱点。例如用基于内容的方法和协同过滤推荐方法各个产生一个推荐预测结果，然后根据某方法组合其结果。

### 2.3.7 推荐算法比较

|  |  |  |
| --- | --- | --- |
| 推荐方法 | 优点 | 缺点 |
| 基于内容推荐 | 推荐结果直观，容易解释；  不需要领域知识 | 新用户问题；  复杂属性不好处理；  要有足够数据构造分类器 |
| 协同过滤推荐 | 新异兴趣发现、不需要领域知识；  随着时间推移性能提高；  推荐个性化、自动化程度高；  能处理复杂的非结构化对象 | 稀疏问题；  可扩展性问题；  新用户问题；  质量取决于历史数据集；  系统开始时推荐质量差； |
| 基于规则推荐 | 能发现新兴趣点；  不要领域知识 | 规则抽取难、耗时；  产品名同义性问题；  个性化程度低； |
| 基于效用推荐 | 无冷开始和稀疏问题；  对用户偏好变化敏感；  能考虑非产品特性 | 用户必须输入效用函数；  推荐是静态的，灵活性差；  属性重叠问题； |
| 基于知识推荐 | 能把用户需求映射到产品上；  能考虑非产品属性 | 知识难获得；  推荐是静态的 |

# 第3章系统需求分析和概要设计

## 3.1 系统需求分析

系统需求分析是确定“为了解决这个问题，目标系统必须做什么“，确定目标系统必须有哪些功能，需求分析阶段确定的系统逻辑模型是以后设计和实现目标系统的基础，也是系统开发人员、开发过程和开发成本等一系列前期需求制定的详细计划。如果没有详细的需求分析，可能在以后开发过程中会出现更改数据库，更改架构甚至重新开始，这样会浪费人力财力。

### 3.1.1 系统结构需求分析

本系统采用当下流行的BS（浏览器\服务器）的结构设计，对于开发和维护来说，只要开发完成，发布在服务器上，维护也在服务端维护就可以，客户端可以说是零维护。对于用户，只需要一个浏览器，就可以访问，不需要像CS一样需要安装客户端，这样方便用户使用。

### 3.1.2 系统功能的需求

通过对当下的视频网站和由推荐功能的网站进行分析，此类系统主要分类两个部分，前台和后台，前台供用户使用，后台供管理使用。前台是用户交互的界面，用户可以注册成为新用户，告诉系统自己的兴趣爱好，系统会根据你的爱好推荐相应的视频，已解决冷启动的问题。观看视频进行评分，不断的训练模型。后台管理员不断的采集新的视频数据，然后添加到系统，供用户观看。

## 3.2 系统的概要设计

### 3.2.1 系统的框架设计

在整个系统的开发当中，分为三层，template层、view层、model层。

Model主要是用来负责业务对象和数据库对象。

Template负责把页面如何展示给用户。

View负责业务逻辑，并在适当的时候调用Model和Template。



### 3.2.2 系统模块的划分

为了更好的管理，开发，维护，本系统划分为以下几个模块，用户模块，推荐模块，日志模块，后台管理模块和定时任务模块。各个模块的功能如下：

用户模块：对用户的登录、注册、查找等功能

推荐功能：实现推荐算法，推荐出用户喜欢的视频

日志模块：收集用户的浏览记录，对视频的评分

后台管理：实现对视频的管理，包括新视频的发布，修改视频的信息，还有用户管理

定时任务：此系统的推荐为离线推荐，因为个人机器的原因，不能实时计算大规模的数据，所以把推荐数据提前算好，定时更新变化的数据。提高系统的性能。



### 3.2.3 用户模块的设计

前台的界面为用户提供的，因此多应该站在用户的角度考虑问题。为了提升用户的体验效果，我们的系统不单单是提供推荐功能，还有队视频做分类，如果可以根据自己的爱好，挑选自己喜欢的类型的视频进行观看。如果用户有很想看的视频，通过搜索可以更精确的找到视频，最后对自己喜欢的视频可以做出评价，系统会根据你的评价给你推荐最符合你的视频。当然为了让系统能记住你，你需要登录到系统。如果你是新用户，不要担心，我们还提供了注册功能，注册的时候还可以告诉系统你喜欢的视频类型。这样就更能推荐出合适的视频。



### 3.2.4 推荐模块的设计

推荐模块当然是系统的最主要的部分，本系统的设计为离线计算（原因见3.2.7定时任务模块），在推荐模块不实现推荐算法，推荐算法由定时任务来实现，此模块对推荐的结果做查询。为了解决冷启动的问题，我们结合多种方式推荐方式，所以在推荐模块就得做控制功能，何时该用什么算法，或者多种算法相结合实现推荐。

### 3.2.5 日志模块的设计

推荐用户最喜欢的视频是系统的关键。那么怎么知道用户喜欢什么？这就得根据用户的浏览记录来判断用户喜欢什么。因此收集用户浏览记录就不可缺少。例如收集看过的视频，观看时间长度，类型，用户观看完后对视频的评分等信息，这就是日志模块该做的事情。

### 3.2.6 管理员的模块设计

如果视频的资源不更新，系统做的再好也不会有大量的用户。大量的用户中当然有非法的用户，因此对视频和用户的管理就是理所当然的事情。管理员模块主要分为视频管理和用户管理。在视频管理中，主要对新视频的增加，修改视频的信息，顶置那些视频做处理。用户管理主要对不合法的用户做禁用等功能。

### 3.2.7 定时任务的模块设计

系统的设计尽可能要有扩展性，一个系统刚开始用户数量可能不多，视频的数量也可能不多，随着时间的推移，这两者都会慢慢的增加。那么如果我们的推荐算法是实时计算的话，可能时间会变长，这样推荐给用户的时间增加，用户在等待推荐结果，体检效果很不佳，因此，在设计的时候就应该考虑这一点。为了避免以上问题的发生。我们采用离线计算推荐结果。系统初始化的时候计算好已有的数据，当有数据变化的时候，更新相应的数据。我们采用定时任务来完成。定时的检验数据是否有变化，有变化则更新变化的数据。这样可以提升系统的性能。

## 3.3 数据库的设计

### 3.3.1 E-R图



### 3.3.2 表的详细设计

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用户表： | | | | |
| 字段名 | 数据类型 | 长度 | 允许空值 | 备注 |
| ID |  |  |  |  |
| user\_name |  |  |  |  |
| Password |  |  |  |  |
| Email |  |  |  |  |
| Hobby |  |  |  |  |
| State |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 视频表： | | | | |
| 字段名 | 数据类型 | 长度 | 允许空值 | 备注 |
| MOVIE\_ID |  |  |  |  |
| TITLE |  |  |  |  |
| YEAR |  |  |  |  |
| GENRES |  |  |  |  |
| DESC |  |  |  |  |
| LINK |  |  |  |  |
| ISNEW |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 评分表： | | | | |
| 字段名 | 数据类型 | 长度 | 允许空值 | 备注 |
| USER\_ID |  |  |  |  |
| MOVIE\_ID |  |  |  |  |
| RATING |  |  |  |  |
| ISUPDATE |  |  |  |  |
| TIMESTAMP |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 相似矩阵： | | | | |
| 字段名 | 数据类型 | 长度 | 允许空值 | 备注 |
| USER\_ID1 |  |  |  |  |
| USER\_ID2 |  |  |  |  |
| VALUE |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 推荐列表： | | | | |
| 字段名 | 数据类型 | 长度 | 允许空值 | 备注 |
| USER\_ID |  |  |  |  |
| MOVIE\_ID |  |  |  |  |
| RECOMMEND\_VALUE |  |  |  |  |

## 3.4 本章小结

# 第4章视频推荐系统的详细设计与实现

## 4.1 系统实现

在上一章中，对系统做了需求分析和设计，清楚了系统该有那些功能，大体的实现有了方向。本章中，从实现方面来做详细叙述。本系统采用Python语言开发，Django Web框架，前台页面主要用HTML ，css，JavaScript，bootstrap等前端框架。数据库采用oracle数据库。本章重点讲在个性化视频推荐系统开发过程的技术问题，编码问题和主要的流程。下面是各个功能之间通过共享数据库之间的联系:



### 4.1.1 Model层的开发

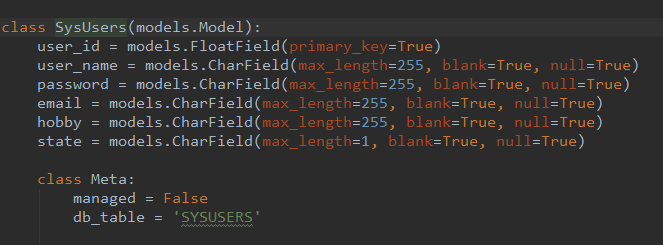
在本层当中主要对数据库中表对象的增加、删除、修改和查询等操作，本层在整个系统中的作用如下：



在数据访问这一层主要封装了对数据库的访问细节，在原始的开发当中，我们如果操作数据库，我们一般都会进行如下的操作，1）创建数据库，设计表结构和字段；2）使用MySQLdb（假设用MySQL数据库）来连接数据库，并编写数据访问层代码（建立连接，编写sql语句，执行sql，关闭连接）；3）业务逻辑层去调用数据访问层执行数据库操作。而在本系统当中，我们使用Django的ORM（Object Relational Mapping，关系对象映射）框架。类和表之间的对应关系如下图；在访问数据库时，只需要用对应的类，调用恰当的API就可以完成。减少代码的编写提高开发效率以及后期的维护。这也是我们选择django框架的原因。



那么，我们的表和类怎么关联起来呢？分为两种情况：第一：根据类创建表，第二根据表创建类。我们先看根据类创建表。如有以下的类（此类需继承Django的Model类），



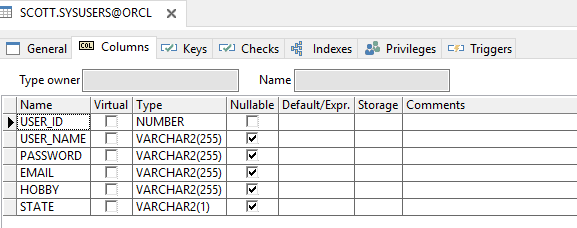
我们执行

python manage.py makemigrations

python manage.py migrate

来进行移植操作，数据库会创建对应的表。

根据表创建类，如果有以下的表

我们执行

python manage.py inspectdb SYSUSERS > apps/user/models.py

生成对应的Model类

### 4.1.2 View层的开发

业务逻辑层是专门处理软件业务需求的一层，处于数据库之上，服务层之下，完成一些列对Domain Object的CRUD，作为一组微服务提供给服务层来组织在暴露给表现层。

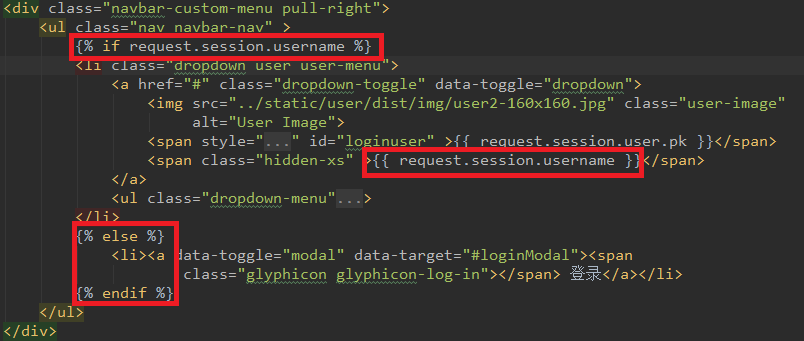
业务逻辑层包含领域对象模型，领域实体，业务规则，验证规则，业务流程。1：领域对象模型为系统结构描述，包含实体功能描述，实体之间的关系。领域模型处于天生的复杂性:2：领域实体：业务层是一些操作业务对象（BO）的处理。业务对象包含数据和行为，是一个完整的业务对象。3:业务最大的逻辑就在处理一些列现实世界的规则，这也是软件中最容易变化的部分，这里通常会出现我们众多的if-else或者switch-case的地方。也这因为如果说以个人觉得在我们的项目最应该关系和分离需求的层次。4：验证规则：业务规则很大程度上也是对对象的数据验证，验证业务对象的当前数据状态。我觉得在每个业务对象上都应该存在一个对外部对象暴露的验证接口。

下面是业务逻辑层在软件结构中的位置：

****

### 4.1.3 Template层的开发

对页面设计进行的任何操作都必须对后台代码进行相应的修改。因此如果可以在不改变后台代码修改的情况下变更设计，那将会方便很多。所以出现了模板的开发。开发模板层时主要包括html+模板语法，模板包括在使用时会被替换掉的变量，和控制模板逻辑的标签。



例如：在用户模块，我们使用if else逻辑标签判断用户是否登录，如登录使用session中的用户替换模板，若没有，显示登录按钮让用户去登录。

## 4.2 功能实现

本小节讲解各个功能的具体的实现。在开发过程中，前台的请求以Ajax的方式请求，后台以json的格式返回数据。Json的格式信息如下：

|  |  |  |
| --- | --- | --- |
| Code | Info | Data |
| 四位数字代码例0000 | “注册成功” | User |

{'code': '0000', 'info': '注册成功', 'data': User }

### 4.2.1用户模块的实现

#### 4.2.1.1 用户注册

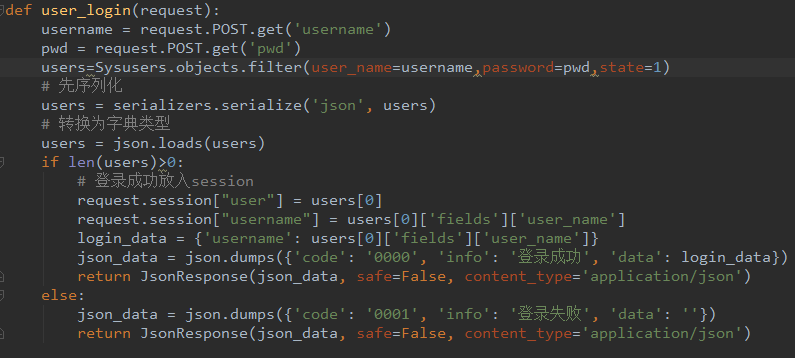
为了能给用户带来更好的体验，推荐符合用户的视频，需要让系统知道你是谁，如果是老用户，已经有账号，直接登录如果没有账号，就是新用户，新用户需要先注册才能登录。下面是新用户注册功能的处理流程。

在系统的首页有登录->注册的按钮，点击进去之后填写相应的信息。填写完成提交之后前后台都会做相应的检验工作，然后判断用户名是否已注册，未注册则给用户生成ID（此ID使用oracle的sequence），把对应的其他信息插入到数据库。则注册成功，用户可以去登录页面登录。

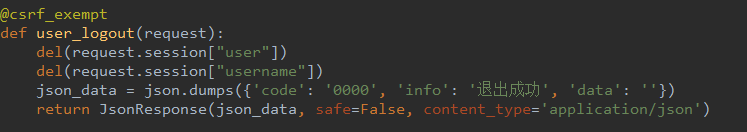
#### 4.2.1.2 用户登录

用户登录的具体流程为：当用户提交登录信息后，前台用javascript做校验，用户名和密码都不能为空，密码不能低于六位。在前台校验成功后，然后封装参数以ajax的方式请求到后台，这样可以减少服务器的压力。后台获取登录的信息后，也必须校验。某些情况下，用户可以不通过前台页面直接请求到后台，因此后台的校验也是必须的。校验通过后再从数据库查找，判断该用户是否存在以及用户名密码是否正确。若存在并且正确，则把用户信息放入session当中，供前台页面显示。否则返回错误提示。

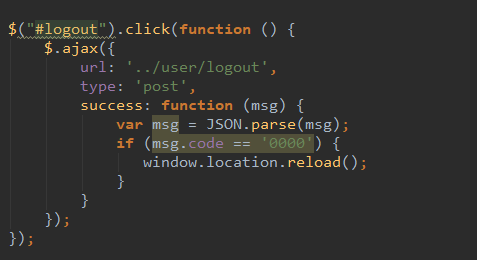
下面是用户登录的后台实现：



用户有登录操作相对用的就有退出操作，用户退出主要是销毁session中用户的信息，界面显示把用户名称更换为登录的按钮。后台的销毁代码如下：



前台点击退出按钮后发送ajax请求到后台，后台返回成功后前台重新加载页面，此时session中已没有用户的信息，所以显示登录的按钮。前台的请求代码：



#### 4.2.1.3 用户其他操作

用户在界面可以进行查找，在用户进行搜索框输入关键字后，点击查找按钮之后，系统会根据输入的关键字按视频的名称进行模糊匹配。然后将查到的视频返回给用户。



界面中还对视频进行分类，在导航栏中有推荐和视频的分类。用户点击某一个分类后台根据类型会为用户查找出相应的类型，为了解决新视频的冷启动问题，如果查出来的视频中含新视频，我们会把新视频顶置，以便用户可以看到新视频。此处都是用户查找操作，因此我们把按分类和用户的搜索抽象成一段公用的逻辑。具体的实现如下：



### 4.2.2推荐模块的实现

推荐模块似乎是我们的此系统的核心模块。根据我们第三章的设计，推荐算法的实现交给我们的定时任务模块。因此这一模块相对简单。我们先看我们的推荐的流程：

1 首先获取session中的用户信息：

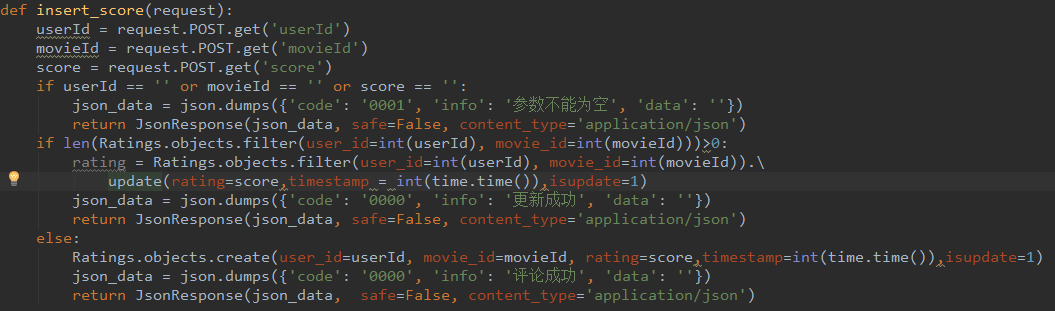
user = request.session["user"]

2 判断是否有用户登录

3 有用户登录按照推荐算法推荐，当为新用户第一次登录的时候，推荐算法也不能推荐，此时推荐出来的数据为0条。这就是用户的冷启动问题，为了解决此问题我们判断是新用户的话，按照用户注册时的爱好信息，根据电影的类型给用户推荐，当用户观看一段时间后，再按照推荐算法推荐。没有用户登录，我们给用户推荐最新的视频，和其他用户对某个视频评分较高的来推荐。

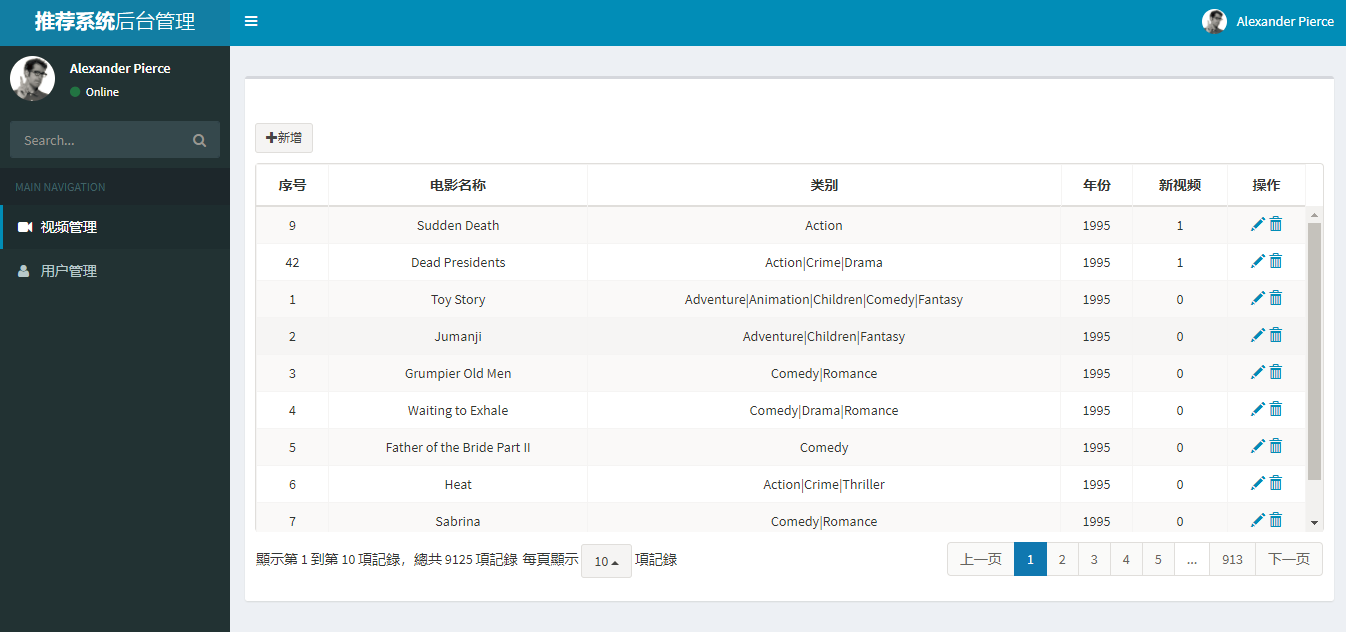
### 4.2.3日志模块的实现

日志模块主要记录用户对看过视频的评分信息，此系统的推荐主要也依赖评分信息。如果用户是第一次给某电影评分，则插入一条记录，此种情况会发生在分类的视频或者查找的视频或者推荐的视频，推荐的视频都是用户未评分的。如果插入的记录记录存在，我们会更新原来的评分。此种情况发生在分类视频和查找的视频当中，因为其中可能有用户看过的视频。因此推荐的类型就掌握在用户的手中，你可以修改评分从而导致推荐结果。下面是后台插入的具体实现：



### 4.2.4管理模块的实现

管理模块主要对视频和用户管理，管理的界面如下：



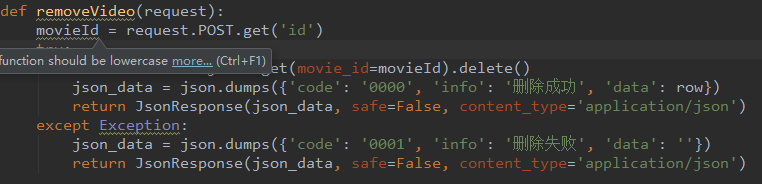
在左边可以切换视频管理和用户管理，我们先看视频管理，在右边会列出所有的视频，我们会把新视频顶置，一行代表一个视频，在每行的最右边有一个操作的栏目，可以对此视频进行删除操作和修改操作。在最上面有一个添加的俺按钮，可以添加新的视频。

视频删除：

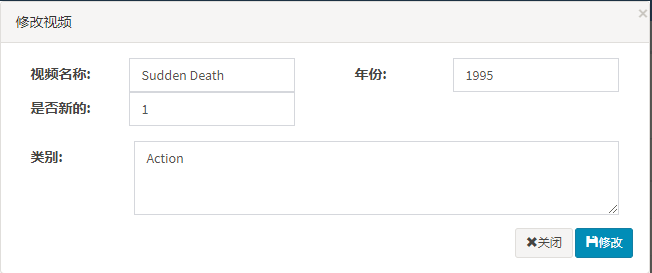
点击删除的时候回给出确认框



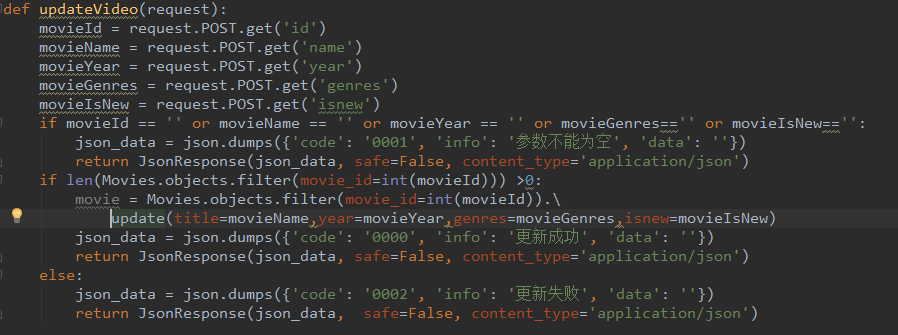
确实是否删除此视频，如果确实此时向后台传视频的id进行删除，取消则不删除。后台删除逻辑实现：



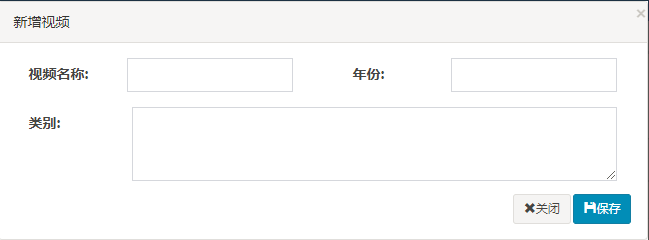
视频修改：点击修改的图标回显此视频的信息如下：



此时可以选择关闭或者更改信息后点击修改按钮。前台把页面的数据传给后台进行修改。后台获取参数，进行校验，校验合法然后修改：



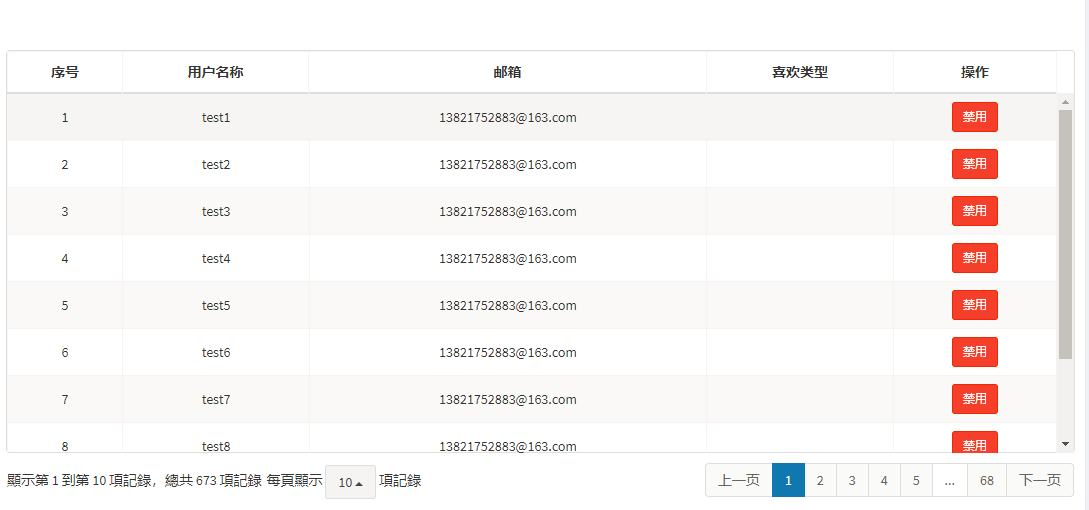
视频增加：点击添加按钮之后会有一个弹出框填写视频的信息：



填写完成后点击保存即可增加新视频：后台也是获取参数，校验，更更新不同的是，这里先查找到视频库的最大的视频的id，然后给此id+1作为新视频的id；

max\_id=Movies.objects.all().order\_by('movie\_id').reverse()[0].movie\_id

下面是用户管理的界面：用户管理比较简单，主要对非法的用户进行禁用，在用户列表的最右边的操作中有禁用的按钮，点击此按钮就可禁用此用户，该用户不可以登录到此系统。



### 4.2.5 定时任务模块的实现

#### 4.2.5.1 基于用户的协同过滤算法

基于用户的协同过滤算法主要包括两个步骤。

（1）找到和目标用户兴趣相似的用户集合。

（2）找到这个集合中的用户喜欢的，且目标用户没有听说过的物品推荐给目标用户。

步骤（1）的关键就是计算两个用户的兴趣相似度。设给定用户u和v，令N(u)表示用户u曾经有过正反馈的物品集合，令N(v)为用户v曾经有过正反馈的物品的集合。通过余弦相似度计算：

下面以用户行为记录为例，举例明UserCF计算用户兴趣相似度的例子：



利用余弦相似度公式计算用户A和用户B的兴趣相似度为

同理，我们可以计算出用户A和用户C、D的相似度：

得到用户之间的兴趣相似度之后，UserCF算法会给用户推荐和他兴趣最相似的K个用户喜欢的物品，如下公式度量了UserCF算法中用户u对物品i的感兴趣度：

其中，包含和用户兴趣最接近的个用户，是对物品有过行为的用户集合，是用户和用户的兴趣相似度，代表用户对物品的兴趣，因为使用的是单一行为的隐反馈数据，所以所有的=1。

#### 4.2.5.2 算法实现

在实现推荐算法的过程中考虑到每次都要从数据库获取到数据，再计算，最后存储到数据库，这样访问数据库有大量的系统资源和时间的消耗，为了解决此问题，我们不采用python语言来实现，而是采用oracle的存储过程。这样大大的提升了计算速度。

相似矩阵的计算：

首先删除数据库中要计算的用户原来的相似矩阵

delete from xsjz x where x.r=userID;

其次遍历所有用户计算两个用户之间的相似度：



FOR j IN (select distinct r.user\_id from ratings r) LOOP

select count(\*)

into fenzi

from (select t.movie\_id

from RATINGS t

where t.user\_id = userID

INTERSECT

select t.movie\_id

from RATINGS t

where t.user\_id = j.user\_id);

select count(\*)

into fenmu1

from RATINGS t

where t.user\_id = userID;

select count(\*)

into fenmu2

from RATINGS t

where t.user\_id = j.user\_id;

xs\_score := fenzi / sqrt(fenmu1 \* fenmu2);

insert into XSJZ (R, C, V) values (userID, j.user\_id, xs\_score);

end LOOP;

commit;

推荐列表的计算

先删除用户的推荐列表：

delete from recommend r where r.user\_id=userID;

commit;

重新计算：

for xsyh in (select \* from xsjz x where x.r=userID order by x.v desc) LOOP

for item in (select \* from ratings ra where ra.user\_id=xsyh.c) LOOP

select count(\*) into isExist from ratings rr where rr.user\_id=userID and rr.movie\_id=item.movie\_id;

*--if isExist>0 说明该电影已经看过 continue*

if isExist=0 then

tmp:=item.rating\*xsyh.v;

select count(\*) into ll from recommend re where re.user\_id=userID and re.movie\_id=item.movie\_id;

if ll=0 then

insert into recommend(user\_id,movie\_id,recommend\_score) values(userID,item.movie\_id,tmp);

commit;

else

select re.recommend\_score into score from recommend re where re.user\_id=userID and re.movie\_id=item.movie\_id;

if tmp>score then

update recommend rec set rec.recommend\_score=tmp where rec.user\_id=userID and rec.movie\_id=item.movie\_id;

commit;

end if;

end if;

end if;

end loop;

end loop;

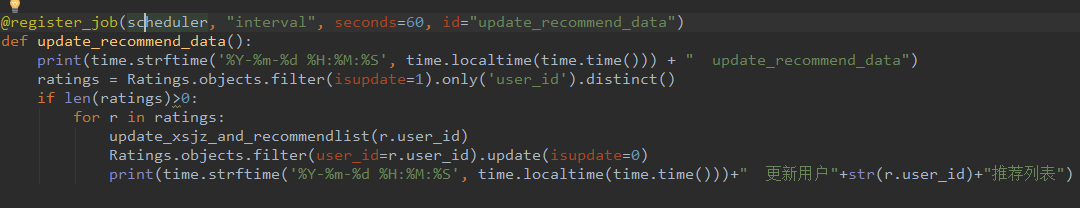
delete from recommend r where r.user\_id=userID and r.recommend\_score=0;

commit;

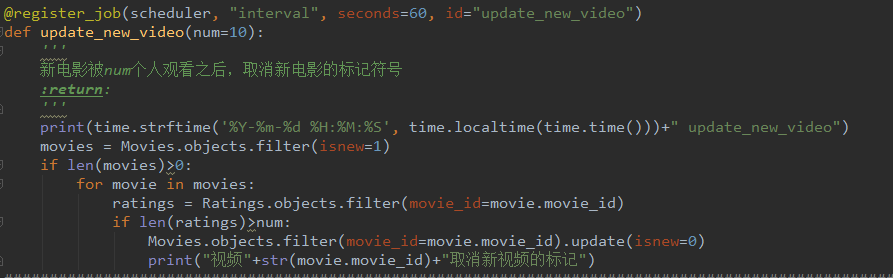
#### 4.2.5.3 定时任务实现

我们采用APScheduler，APScheduler是基于Quartz的一个python定时任务框架，实现了Quartz的所有功能，使用起来十分方便。提供了基于日期、固定时间间隔以及crontab类型的任务，并且可以持久化任务。

我们的定时任务不断的判断是否要更新推荐列表，当用户对某个视频的评分改变之后，这条记录在数据库表示为需要更新，定时任务查询到后调用存储过程来更新该用户的推荐列表。最后更新该条记录为不再需要更新的状态。下面是我们的python代码实现逻辑。



定时任务还需要实现另一个功能，当新视频上架之后，什么情况下不是新视频。这个功能我们也用定时任务来完成，规定当超过十个用户观看之后取消新视频的标志。下面是具体的实现。



## 4.3 本章小结

本章从系统的实现方面进行了讲解，对复杂的逻辑给出了流程图，每个实现还有文字描述和部分代码。

# 第5章个性化视频推荐系统测试

## 5.1 测试环境

## 5.2 功能测试

## 5.3 用户界面测试

## 5.4 后台管理测试

## 5.5 性能测试

# 总结与展望

# 参考文献

# 致谢