2020-7-14

《挑战极限报告》

——NotOnlyFilm电影垂直搜索引擎

**《项目实训》2班 G23 组长：张佳瑶 组员：贺婷婷 应承峻 戴陈威 杨建伟**

目录

[1 挑战极限简介 2](#_Toc45630327)

[2 深度搜索 3](#_Toc45630328)

[2.1 查询分发与结果整合 3](#_Toc45630329)

[2.2 多维排序 4](#_Toc45630330)

[2.3 规则搜索 9](#_Toc45630331)

[2.4 智能补全 13](#_Toc45630332)

[2.5 搜索指数可视化 14](#_Toc45630333)

[3 深度分析 16](#_Toc45630334)

[3.1 影评关键字词云 16](#_Toc45630335)

[3.2 同类型电影对比 17](#_Toc45630336)

[3.3 猜你喜欢推荐 18](#_Toc45630337)

[3.4 基于知识图谱的自动问答 19](#_Toc45630338)

[3.5 关系图谱可视化 22](#_Toc45630339)

# 挑战极限简介

在十五天的项目实训课程实践过程中，团队开发完成了一款名为Not Only Film的电影领域垂直搜索引擎。除了完成搜索关键字得到相关结果并予以显示的基本功能外，团队对于极限挑战进行了一定的设计与实现，扩展了平台的功能，提升了用户的使用体验。

在深度搜索部分，团队开发完成了查询分发与结果整合、多维排序、规则搜索、智能补全、搜索指数可视化的功能；在深度分析方面，团队实现了影评关键字词云、同类型电影对比、猜你喜欢推荐、基于知识图谱的自动问答、关系图谱可视化的功能。

# 深度搜索

## 查询分发与结果整合

### 简介

在搜索时能够分发查询关键字，在电影、资讯、种子三种不同类型的数据资源中进行搜索并返回符合关键字的结果。

在搜索结果页默认“全部”展示，聚合所有类型的数据资源，依据内容相关度以及数据资源类型排序展示。

### 原理及实现

使用Elastic Search中的多索引查询结合MultiQuery多字段匹配实现，为不同索引的不同字段分配不同的权重进行查询，将返回的结果使用泛型数组List<QueryResBo>存储，所有的返回结果类MovieResBo, RecommendResBo, MagnetResBo，MagnetResBo全部继承QueryResBo基类。

### 效果展示



图1 多类型资源整合后的搜索结果-1



图2 多类型资源整合后的搜索结果-2

## 多维排序

### 简介

能够针对某一特定类别的资源进行多种类型的排序，默认采用相关性从高到低的排序方式，可选评分从高到低、时间从近到远的排序方式。

支持对于电影类型数据按类型、地区、年份、时长进行筛选，允许设置多个筛选条件；支持对于资讯类型数据按发布时间进行筛选。

### 原理及实现

筛选时，通过构建Elastic Search中BoolQuery的Filter，对于单值筛选可以使用term设置筛选条件，而对于多值筛选可以使用terms设置筛选条件。

对于排序，直接指定Elastic Search中Sort进行字段配置。此外通过排序重打分的方式来牺牲排序的绝对性换取准确度的提升，以按时间从近到远的排序为例，

在使用排序的功能时，为了达到严格的排序效果，往往会严重降低查询的准确性，而这往往很难满足用户需要，因此我们需要对排序效果做出一定的牺牲，来换取查询结果的准确性。即我们在对时间进行排序时，不是严格地按照时间进行排序，而是根据其匹配的分值得分×时间衰减率，进行重打分计算

时间打分模型：通过在ScriptSort引入指数衰减曲线和Gauss衰减函数曲线分别对评分和时间进行衰减处理。在Gauss衰减函数曲线中，我们根据年份的分布情况对下图中f(d)做了微调，经过多次调参我们确定了模型函数。

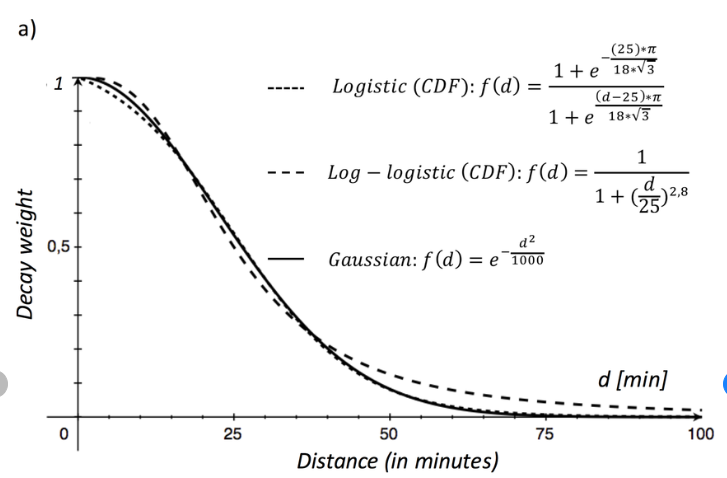


图3 衰减曲线

### 效果展示

#### 多维排序

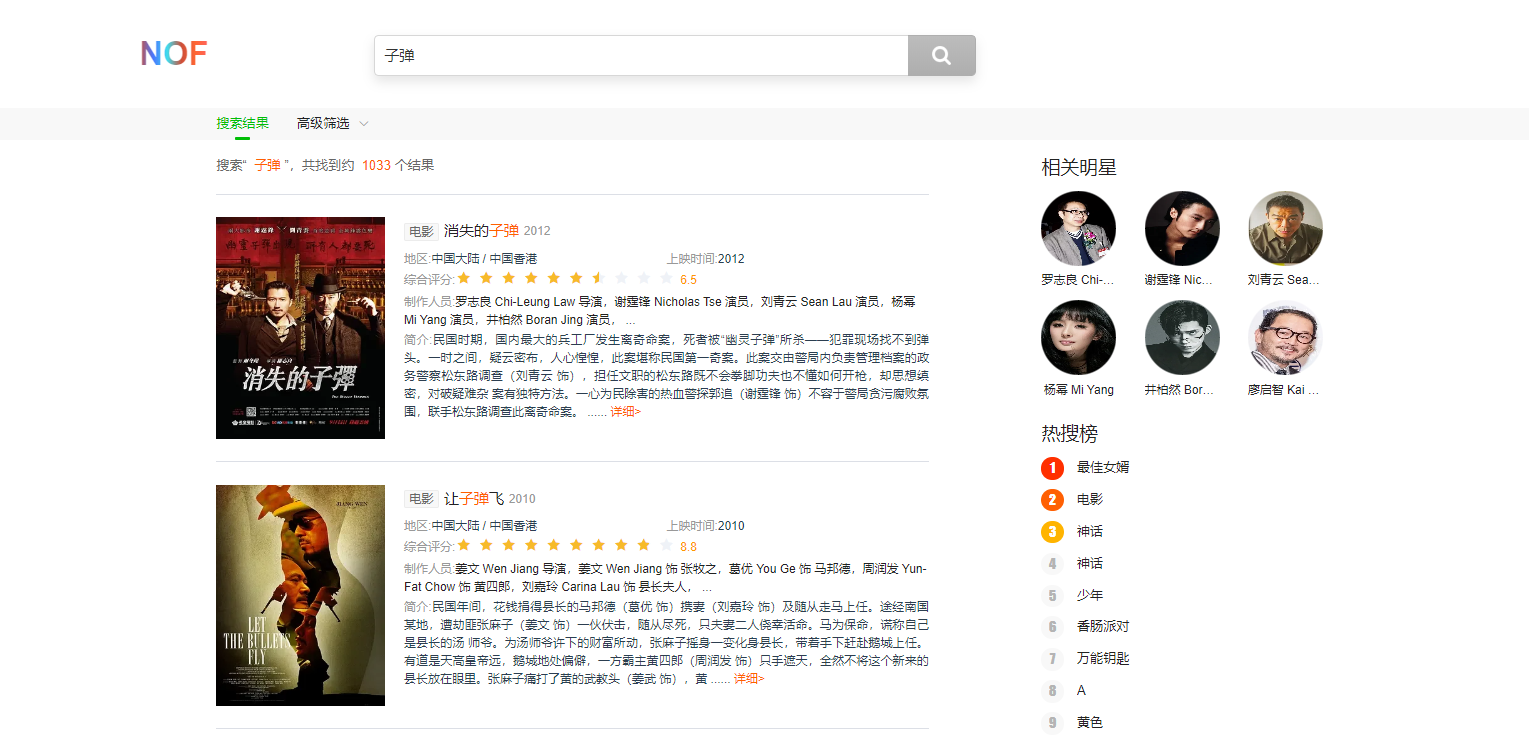


图4 默认按照相关性排序

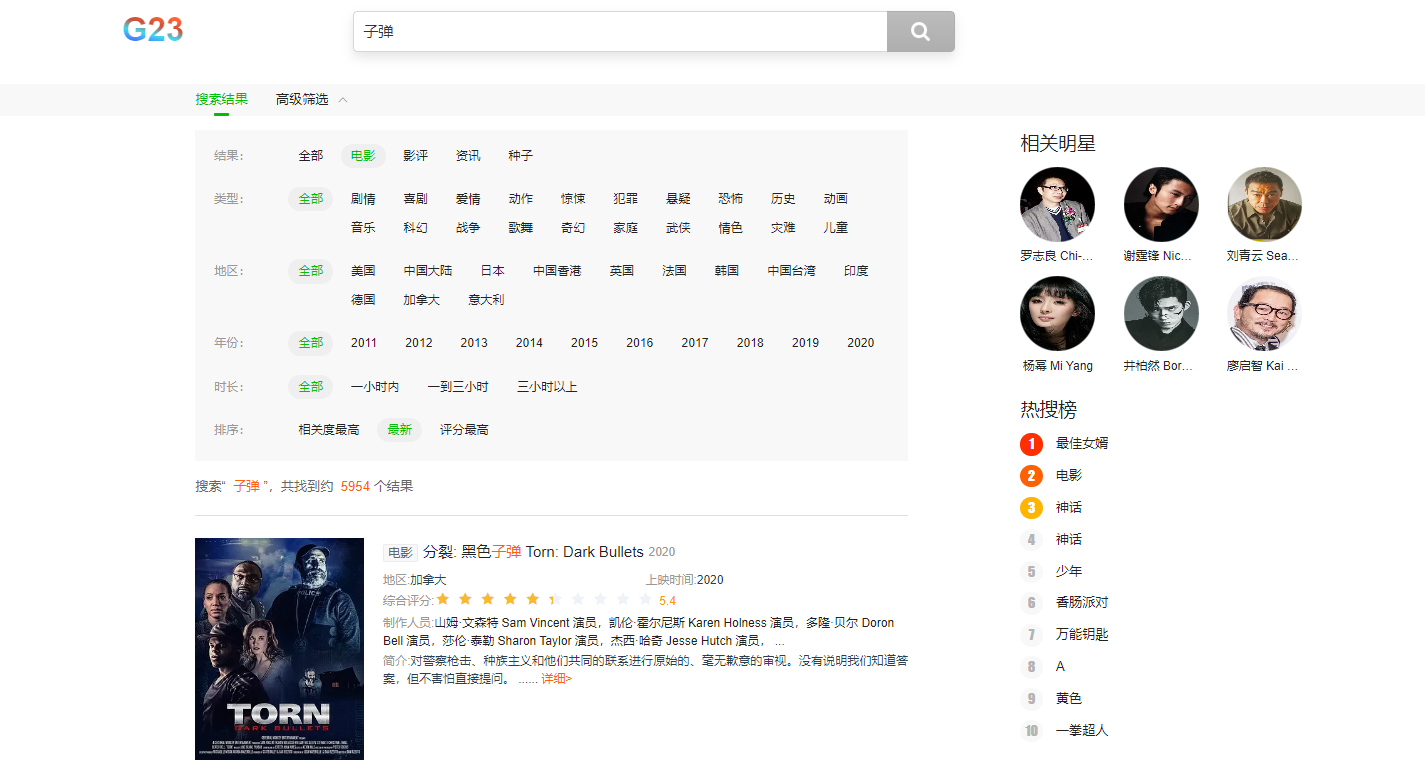


图5 点击最新采用从远到近排序

#### 多维筛选



图6 默认相关性排序电影类型全部结果



图7 多维筛选后得到符合要求的结果

## 规则搜索

### 简介

用户可以根据搜索符号规则组织搜索输入语句，进行搜索范围限定，获得更加精准的搜索结果。

Not Only Film平台设计实现了以下四种高级规则搜索：

* “+<Key>”表示Key必须在查询结果中出现。
* “-<Key>”表示Key不应该在查询结果中出现。
* “<Key1> <Key2>”表示表示两个关键字匹配任意一个均可。
* “:intitle <Key>”表示匹配的查询结果的标题中必须要出现Key。

### 原理及实现

根据通用的查询语法，“+<Key>”表示这个关键字必须在查询结果中出现，“-<Key>”表示这个字段不应该在查询结果中出现，“<Key1> <Key2>”表示表示两个关键字匹配任意一个均可，“:intitle <Key>”表示匹配的查询结果的标题中必须要出现这个关键字。

我们设计了一种通用的查询结构：使用ES的布尔查询加上排序选项以及高亮选项。分析可得，布尔查询主要由与（Must）、或（Should）、非（Must Not）以及一个过滤器组成。

因此，Must的作用主要是匹配查询字符串中的“+”符号， Should字符串主要匹配查询字符串中的空格，Must Not主要匹配查询字符串中的“-”符号。

{

"query": {

"bool": {

"must": [],

"should": [],

"filter": [],

"must\_not": []

}

},

"sort": [],

"highlight": {

"pre\_tags": "<em>",

"post\_tags": "</em>",

"fields": {}

},

"from": 0,

"size": 10

}

### 效果展示

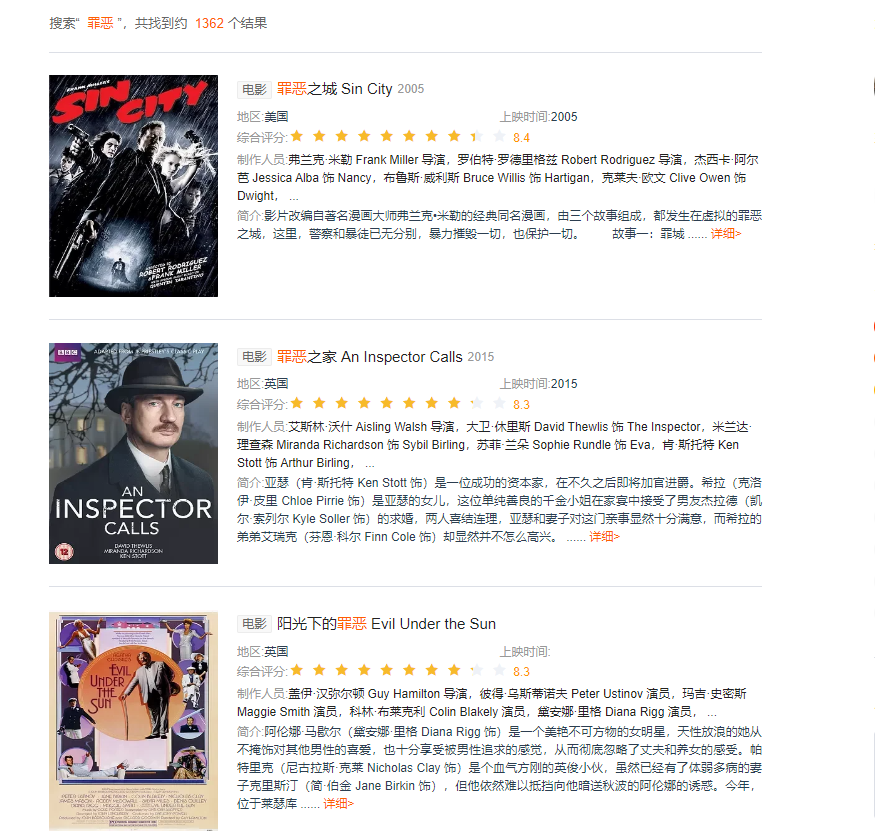


图8 “罪恶”搜索结果

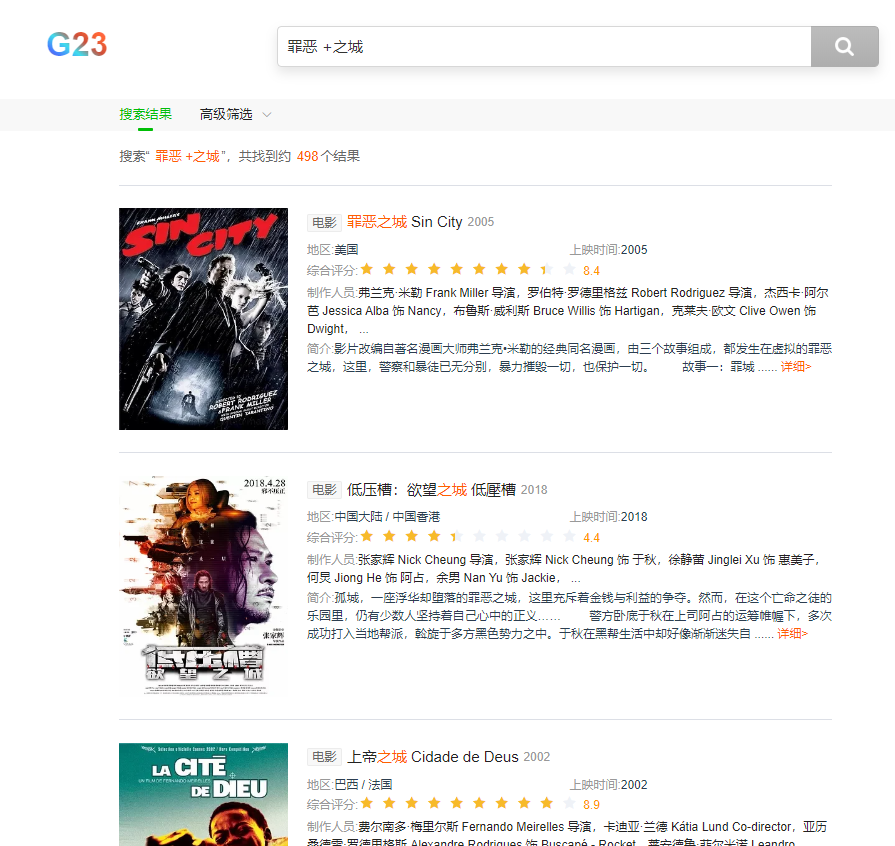


图9 “罪恶 +之城”搜索结果

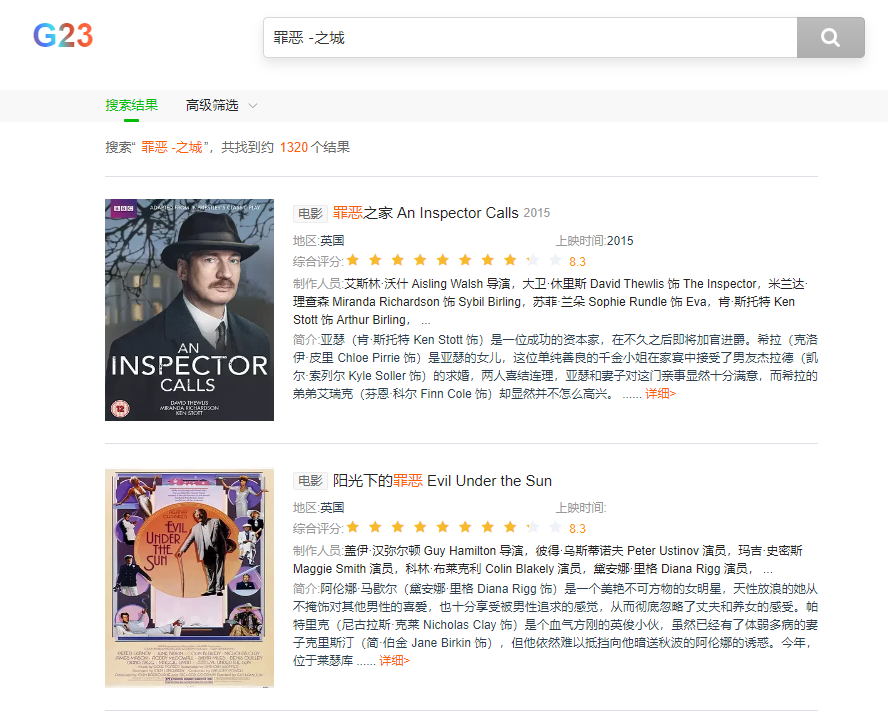


图10 “罪恶 -之城”搜索结果

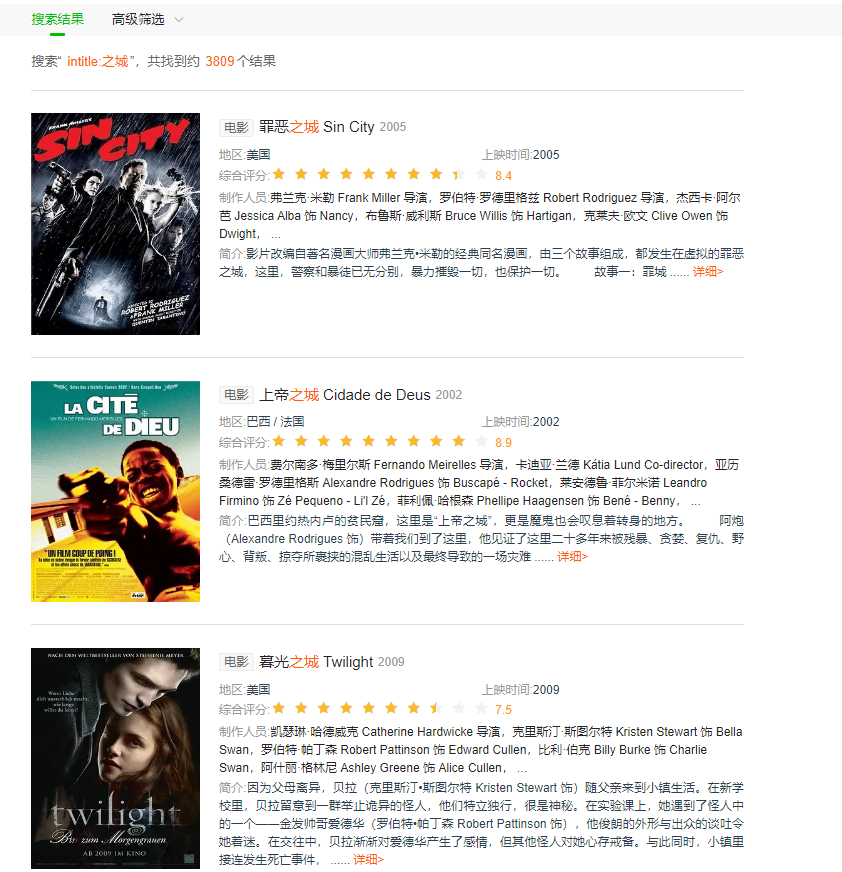


图11 “intitle:之城”搜索结果

## 智能补全

### 简介

在搜索框中根据用户已输入的文字，提示智能补全后的结果。

### 原理及实现

为了实现在搜索框中输入前缀自动补全的功能，有很多可选的实现方案。其中最常见的就是使用正则匹配或是PrefixQuery，即使用key\*的形式来匹配关键词，但是这样往往要对索引进行全文搜索，其搜索效率在数据量大的时候非常低。

为了提高搜索效率，我们使用了ES中的Completion Suggester进行前缀匹配，它通过了改造索引的结构来提升性能。在索引结构的title字段中，与前面的拼音一样，我们新加一个suggest域，并设置类型为completion（经过测试研究，我们发现不能对title直接进行completion类型的设置，它的分词会导致该类型消耗约20倍的存储空间）。Completion Suggester与ES中提供的其他Suggester相比，采用了不同的数据结构，索引并非通过倒排来完成，而是将分析过的数据编码成FST和索引一起存放，FST有着非常高的数据压缩率和查询效率。对于一个开放状态的索引，FST会被ES整个装载到内存里的，能够进行极快速度的前缀查找。

### 效果展示

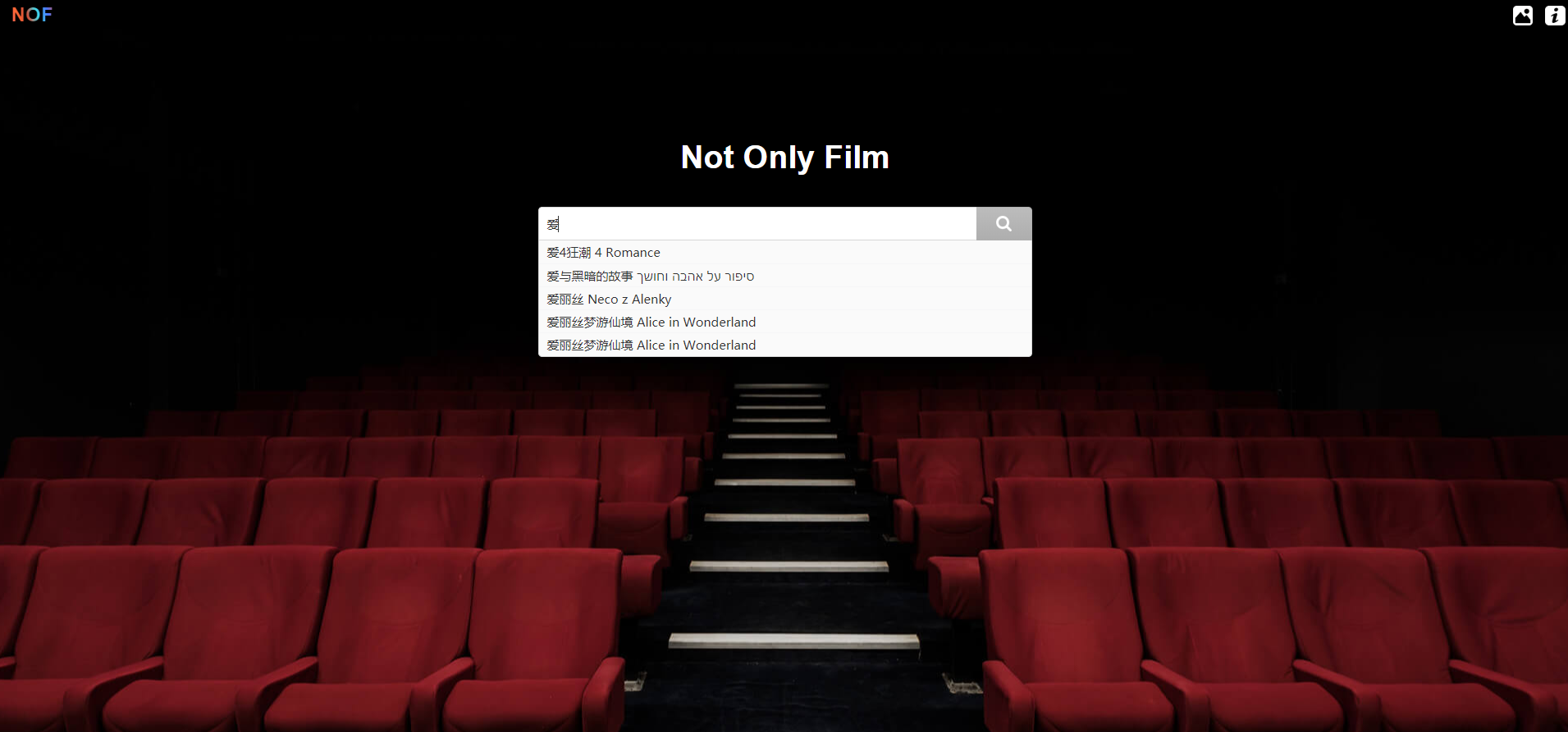


图12 搜索框智能补全效果

## 搜索指数可视化

### 简介

通过折线图的形式展示最近7天的搜索指数，表现电影的热度趋势。

### 原理及实现

由于网站用户较少，目前采用爬取百度指数的方式获得搜索指数的数据。在第一次爬取数据时爬取近14天的数据，然后在之后的日子里每天启动一次脚本爬取前一天的数据，从而保证数据的时效性。

### 效果展示



图13 近7天搜索指数折线图

# 深度分析

## 影评关键字词云

### 简介

针对电影相关的影评进行关键词提取和词云可视化。

### 原理及实现

利用爬取到的电影相关影评，进行分词、去停用词操作，将内容分割至单词维度。按照单词出现的频率排序，取高频词语作为关键词，设定最大、最小阈值及字体大小、颜色排布，生成词云对影评关键词进行展示，从而可视化表示用户对于该影片的评分。

### 效果展示



图14 影评关键字词云

## 同类型电影对比

### 简介

对于一部电影，将其与同类型电影进行对比，得到不同维度上的数据，表现该类影片在同类型影片中的地位。

### 原理及实现

对于电影A，其类型为B，选择所有类型为B的电影，在维度C上进行升序排序，取得电影A的排名后计算其排名在同类电影中的分布位置，得到电影A好于“电影A排名/B类型电影总数” B类型片的数据分析结论。

按照如上逻辑，从电影评分、评分人数、近期热度三个维度对一个电影所属的所有类型进行同类推理得到分析结论。

* 电影评分：由于系统同时爬取了IMDB评分和豆瓣评分，所以在计算电影评分时采用评分人数进行加权后得到电影评分，计算公式为：
* 评分人数：由于系统同时爬取了IMDB评分和豆瓣评分，所以在计算电影评分人数时采用评分人数之和，计算公式为：
* 近期热度：取该电影近14日搜索指数的平均，计算公式为：

### 效果展示



图15 同类型电影对比

## 猜你喜欢推荐

### 简介

系统提供推荐功能，根据用户的访问记录、电影的详细信息向用户推荐用户可能喜欢的电影。

### 原理及实现

推荐的参考依据：(1)近10次的电影访问记录；(2)同类型的电影；(3)相同电影搜索记录的用户搜索过的其他电影。综合三个方面的信息，最终为用户推荐他可能喜欢的电影。推荐电影展现在电影详情页，根据用户实时的访问记录，动态地推荐相似类型的电影。

### 效果展示



图16 猜你喜欢

## 基于知识图谱的自动问答

### 简介

本系统构建基于知识图谱的自动问答，面向电影领域回答相关问题。

例如，演员的生日、出生地、英文名、出演的电影、合作的演员等信息，电影的上映日期、评分、类型等信息。

### 原理及实现

#### 知识图谱本体建模

本系统为面向电影领域的垂直搜索引擎，涉及的概念和范围是固定可控的，数据精度较高，因此构建领域知识图谱。本图谱是一个电影领域的知识图谱，采用自顶向下的方法来构建本体。

在Protégé中创建电影知识图谱的类。电影知识图谱中的类有电影、电影类型、电影人物等。其中电影、电影类型、电影人物是互斥的关系。电影人物下根据不同的派别，有子类如喜剧演员等。

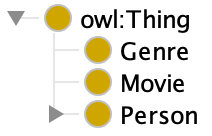


图17 类

在Protégé中创建类的对象属性。“hasActedIn”表示某电影人物参演了某电影，“hasActor”表示某电影有哪些电影人物，“hasGenre”表示某电影有哪些电影类型。“hasActedIn”属于电影人物类，取值范围是Movie，它的逆属性是“hasActor”。

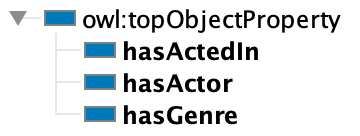


图18 对象属性

在Protégé中创建类的数据属性。主要的数据属性有电影类型名字、电影简介、电影评分、电影上映日期、电影人物名字、电影人物生平等等。

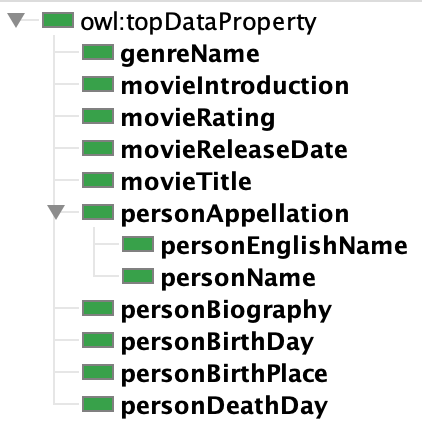


图19 数据属性

最后建立完成的本体结构为：

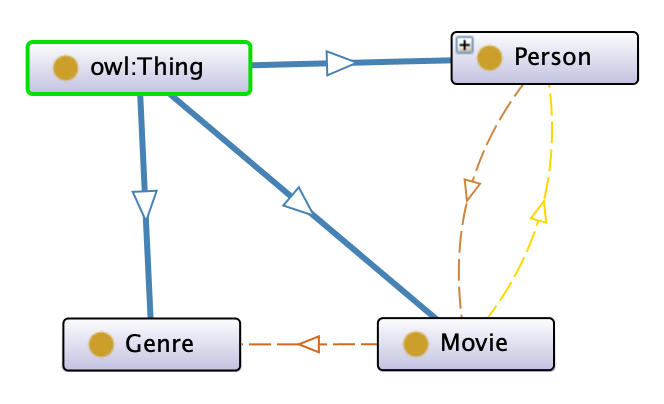


图20 本体结构

#### 数据准备

将从豆瓣和TMDb网站爬取获得的信息存入到关系数据库MySQL中。演员的基本信息包括姓名、出生日期、生平等。电影的基本信息包括名称、简介、评分、上映日期、类型等。采用数据库到三元组的直接映射的方式，使用D2RQ、Jena将MySQL数据库中的数据转换为三元组，存入Fuseki数据库中。

在Fuseki中存储准备好的三元组数据，并且配置配置文件，将我们设计好的本体文件放入Fuseki数据库的配置文件中，启动OWL推理机。从推理机中可以推理得到原本数据中未知的信息，如一部电影有哪些演员出演，某个演员一共演出了多少部电影。

#### KBQA

利用构建的电影知识图谱实现问答。由于面向电影行业，我们构建电影人物名字和电影名字的外部词典。使用jieba并且加载外部词典配合正则匹配完成分词和实体识别。使用REfO完成语义匹配，匹配完成后使用SPARQL向Fuseki服务器发送查询，打印最终结果。最终实现电影领域的简易的KBQA。

### 效果展示



图21 KBQA

## 关系图谱可视化

### 简介

为了提供用户更好、更直观的搜索体验，我们将电影详情、电影类型、电影人关系可视化，本系统构建了全部电影和单独电影的关系图谱。

### 原理及实现

从数据爬取中获得全部电影的信息，解析出合适的关系范围。对于全部电影，构建全部电影之间的关系。考虑到计算机渲染，本系统为每个电影解析适当大小的关系图谱，围绕一个电影构建该电影的电影类型、电影人与电影本身的关系，同时再添加电影人出演的其他电影的关系，一共是两层的关系。

关系范围用JSON格式存储，借助开源工具InteractiveGraph，使用JSON绘制相应的关系图谱。

### 效果展示

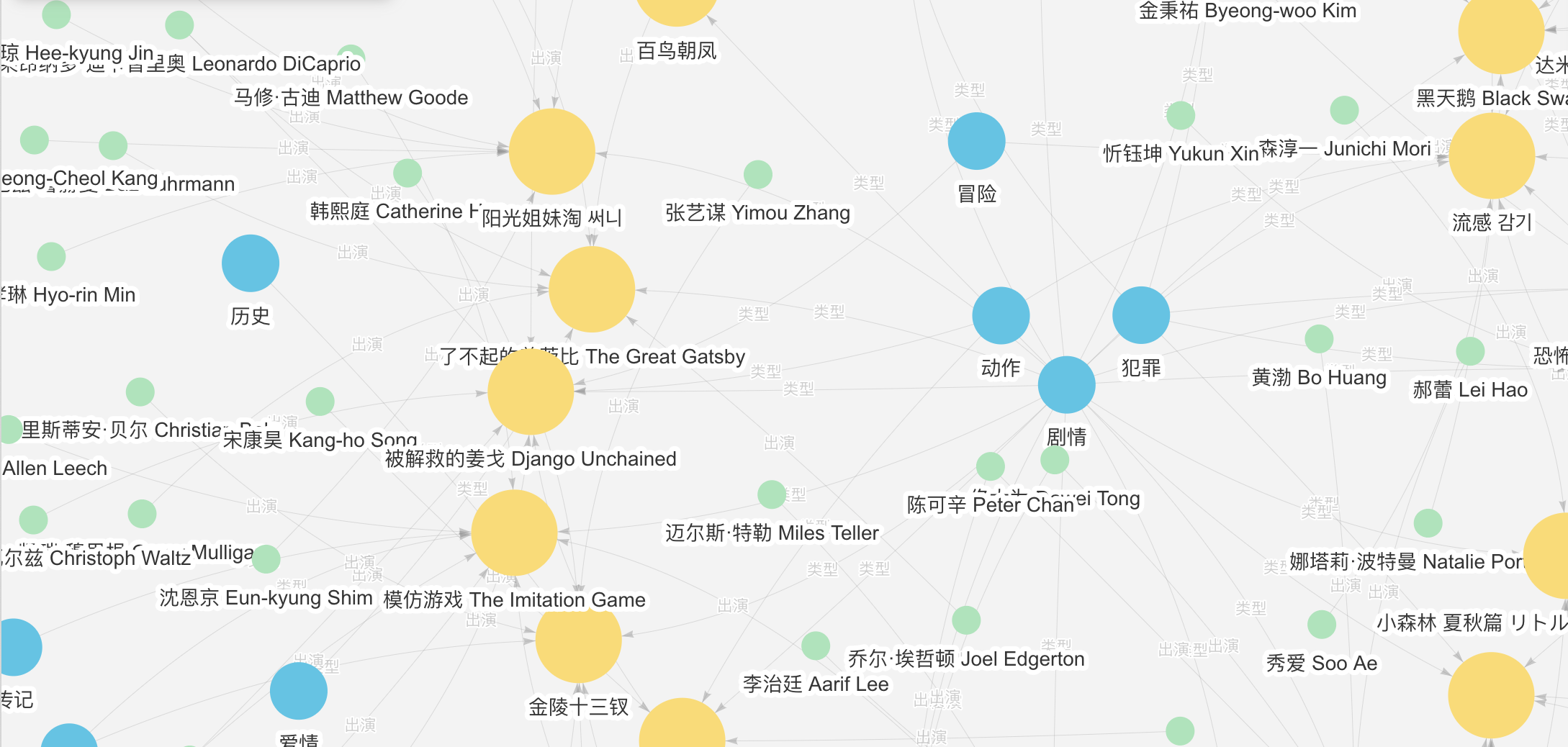


图22 全部电影关系图谱

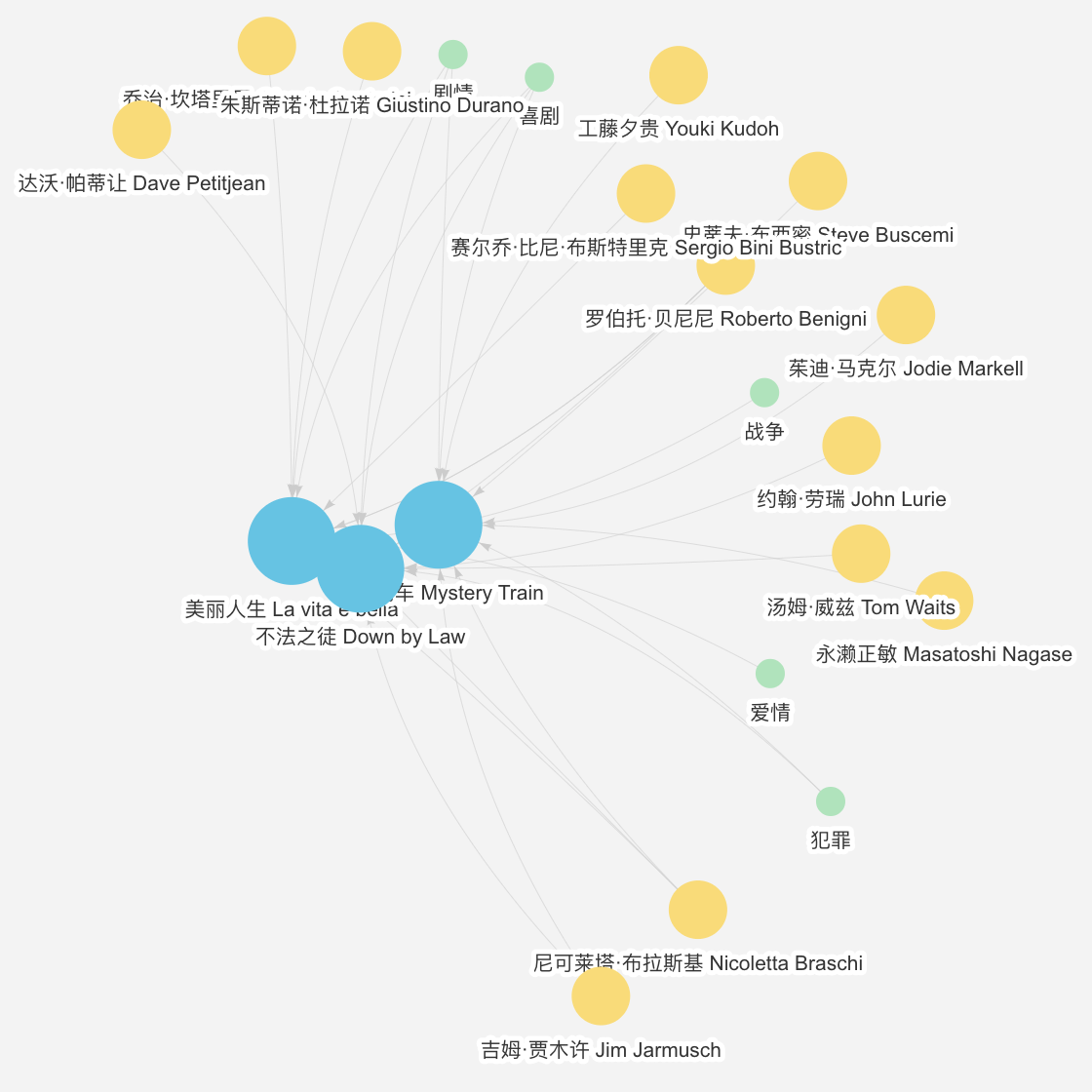


图23 美丽人生的关系图谱