****



**研 究 生 毕 业 论 文**

**（申请工程硕士学位）**

|  |  |
| --- | --- |
| **论文题目** | 裁判文书检索与推荐系统  的设计与实现 |
| **作者姓名** | 朱鹏 |
| **学科、专业名称** | 工程硕士(软件工程方向) |
| **研究方向** | 软件工程 |
| **指导教师** | 葛季栋 副教授 |

**2019年 4 月 30 日**

**学 号： MF1732201**

**论文答辩日期： 2019 年 05 月 23 日**

**指 导 教 师： （签字）**

**裁判文书检索与推荐系统的设计与实现**

|  |  |
| --- | --- |
| **作 者:** | **朱鹏** |
| 指导教师: | **葛季栋　副教授** |

|  |
| --- |
| **南京大学研究生毕业论文** |
| **(申请工程硕士学位)** |

|  |
| --- |
| **南京大学软件学院** |
| 2019**年**04**月** |

**The Design and Implementation of Judgment Documents Retrieval and Recommendation System**

**Zhu, Peng**

**Submitted in partial fulfillment of the requirements for the degree of Master of Engineering**

Supervised by

Associate Professor **Ge, Jidong**

Software Institute

**NANJING UNIVERSITY**

Nanjing, China

April, 2019

# 摘 要

近些年来，随着互联网科技的快速发展，推进法院信息化的建设工作也被纳入战略部署。伴随着数据以指数级别的速度累积增长，以及大数据技术的日渐成熟，基于司法大数据的一系列研究都相继展开。裁判文书是诉讼活动结果的载体，记载着人民法院审理过程和结果，也是开展法院大数据研究工作的重要基础。在人工智能的研究背景下，任何实验的研究与验证都离不开大量有效文书数据的支持。对此，通过裁判文书检索与推荐系统筛选出相关性文书集数据是开展一系列研究的重要前提与基础。

本文基于裁判文书的检索方式，设计并实现了裁判文书检索与推荐系统。系统分为四个功能模块，包括有文书处理模块、文书管理模块、文书检索模块以及文书推荐模块。其中前两个功能模块主要针对文书数据进行处理，包括文书解析，将xml文书转换成对应的doc文书，构建文书索引以及文书信息的更新。后面两个功能模块主要是实现用户的功能性需求，能够根据文书条件以及文书内容进行检索并推荐相似的文书。

在内容结构上，本文首先介绍了裁判文书检索与推荐系统的项目背景，阐述了本文的主要工作，从宏观上对本文的组织结构进行了介绍。紧接着介绍了本系统所使用的相关主流技术，包括MVC的设计思想、Spring、Spring MVC、Elasticsearch搜索引擎以及LDA主题模型。然后通过对项目功能需求的分析，介绍了系统相关模块与数据库的设计，并且通过相关视图分析了系统的架构设计。接着基于系统的需求分析以及总体设计，重点讲述了四个功能模块的具体实现方案以及相关的实现效果。最后对系统进行性能测试，确保用户操作的响应时间在规定需求以内。

**关键词：**裁判文书，MVC，Spring，Elasticsearch，LDA

# Abstract

In recent years, with the rapid development of Internet technology, the information construction of court has been included in the strategic deployment. Along with the cumulative growth of data at an exponential rate and the maturity of big data technologies, a series of studies based on judicial big data have been launched. Judging documents are the carrier of the results of litigation activities, documenting the trial process and results of the people's courts, and also an important basis for carrying out the research work of the court's big data. In the context of artificial intelligence research, the research and verification of any experiment is inseparable from the support of a large number of valid paper data. In this regard, screening the relevant document set data through the judgment document search and recommendation system is an important premise and basis for conducting a series of studies.

Based on the retrieval method of the judgment documents, this thesis designs and implements the retrieval and recommendation system of the judgment documents. The system is divided into four functional modules, including a document processing module, a document management module, a document retrieval module, and a document recommendation module. The first two functional modules are mainly processed for the document data, including document analysis, converting the xml document into the corresponding doc document, constructing the document index and updating the document information. The latter two functional modules are mainly to realize the functional requirements of the user, and can search and recommend similar documents according to the conditions of the documents and the contents of the documents.

In terms of content structure, this thesis first introduces the project background of the judgment document retrieval and recommendation system, expounds the main work of this thesis, and introduces the organizational structure of this thesis from a macro perspective. Then it introduces the relevant mainstream technologies used in this system, including MVC design ideas, Spring, Spring MVC, Elasticsearch search engine and LDA theme model. Then through the analysis of the project's functional requirements, the design of the system-related modules and database is introduced, and the architecture design of the system is analyzed through related views. Then based on the system's requirements analysis and overall design, the specific implementation of the four functional modules and related implementation effects are highlighted. Finally, the system is tested for performance to ensure that the response time of the user's operation is within the specified requirements.

**Keyword:** Judgment documents, MVC, Spring, Elasticsearch, LDA

# 目 录

[摘 要 I](#_Toc10184991)

[Abstract II](#_Toc10184992)

[目 录 IV](#_Toc10184993)

[图目录 VI](#_Toc10184994)

[表目录 VIII](#_Toc10184995)

[第一章 引言 1](#_Toc10184996)

[1.1 项目背景 1](#_Toc10184997)

[1.2 国内外研究现状 2](#_Toc10184998)

[1.3 本文主要研究工作 4](#_Toc10184999)

[1.4 本文的组织结构 5](#_Toc10185000)

[第二章 技术综述 6](#_Toc10185001)

[2.1 MVC模式 6](#_Toc10185002)

[2.1.1 MVC模式的简介 6](#_Toc10185003)

[2.1.2 MVC模式的优缺点 7](#_Toc10185004)

[2.2 Elasticsearch搜索引擎 7](#_Toc10185005)

[2.2.1 Elasticsearch简介 7](#_Toc10185006)

[2.2.2 Elasticsearch基本概念 8](#_Toc10185007)

[2.2.3 Elasticsearch架构 9](#_Toc10185008)

[2.3 LDA主题模型 10](#_Toc10185009)

[2.3.1 LDA主题模型简介 10](#_Toc10185010)

[2.3.2 LDA主题模型的原理 10](#_Toc10185011)

[2.4 Spring框架 12](#_Toc10185012)

[2.4.1 Spring介绍 12](#_Toc10185013)

[2.4.2 Spring MVC介绍 13](#_Toc10185014)

[2.5 本章小结 14](#_Toc10185015)

[第三章 裁判文书检索与推荐系统的分析与设计 15](#_Toc10185016)

[3.1 系统概述 15](#_Toc10185017)

[3.2 系统需求分析 15](#_Toc10185018)

[3.2.1 系统用例 15](#_Toc10185019)

[3.2.2 功能需求 16](#_Toc10185020)

[3.2.3 非功能需求 19](#_Toc10185021)

[3.3 系统总体设计 20](#_Toc10185022)

[3.3.1 系统模块划分 20](#_Toc10185023)

[3.3.2 系统逻辑视图 22](#_Toc10185024)

[3.3.3 系统开发视图 23](#_Toc10185025)

[3.3.4 系统物理视图 23](#_Toc10185026)

[3.3.5 总体结构 24](#_Toc10185027)

[3.4 系统模块设计 25](#_Toc10185028)

[3.4.1 文书处理模块 25](#_Toc10185029)

[3.4.2 文书管理模块 27](#_Toc10185030)

[3.4.3 文书检索模块 29](#_Toc10185031)

[3.4.4 文书推荐模块 32](#_Toc10185032)

[3.5 数据库设计 35](#_Toc10185033)

[3.6 本章小结 37](#_Toc10185034)

[第四章 裁判文书检索与推荐系统的实现 38](#_Toc10185035)

[4.1 裁判文书检索与推荐系统的主要功能概述 38](#_Toc10185036)

[4.2 文书处理模块的实现 39](#_Toc10185037)

[4.3 文书管理模块的实现 43](#_Toc10185038)

[4.4 文书检索模块的实现 44](#_Toc10185039)

[4.5 文书推荐模块的实现 48](#_Toc10185040)

[4.5.1 LDA主题模型的训练 49](#_Toc10185041)

[4.5.2 相似性文书推荐 51](#_Toc10185042)

[4.5.3 推荐效果评估 52](#_Toc10185043)

[4.6 裁判文书检索与推荐系统的测试 56](#_Toc10185044)

[4.6.1 测试思路 56](#_Toc10185045)

[4.6.2 性能测试 56](#_Toc10185046)

[4.7 本章小结 58](#_Toc10185047)

[第五章 总结与展望 59](#_Toc10185048)

[5.1 总结 59](#_Toc10185049)

[5.2 进一步工作展望 60](#_Toc10185050)

[参考文献 61](#_Toc10185051)

[致 谢 64](#_Toc10185052)

[版权及论文原创性说明 65](#_Toc10185053)

# 图目录

[图2.1 Elasticsearch体系架构 9](#_Toc10185054)

[图2.2 LDA概率图 11](#_Toc10185055)

[图2.3 Spring框架图 13](#_Toc10185056)

[图2.4 Spring MVC工作流程图 14](#_Toc10185057)

[图3.1 系统用例图 16](#_Toc10185058)

[图3.2 系统功能模块图 21](#_Toc10185059)

[图3.3 系统逻辑视图 22](#_Toc10185060)

[图3.4 系统开发视图 23](#_Toc10185061)

[图3.5 系统物理视图 24](#_Toc10185062)

[图3.6 系统架构图 25](#_Toc10185063)

[图3.7 文书解析流程图 26](#_Toc10185064)

[图3.8 文书类型转换数据流图 27](#_Toc10185065)

[图3.9 文书索引过程图 28](#_Toc10185066)

[图3.10 文书更新流程 28](#_Toc10185067)

[图3.11 高级检索序列图 29](#_Toc10185068)

[图3.12 高级检索类图 30](#_Toc10185069)

[图3.13 文书下载序列图 31](#_Toc10185070)

[图3.14 文书收藏流程图 32](#_Toc10185071)

[图3.15 文书推荐框架总览图 33](#_Toc10185072)

[图3.16 文书推荐流程图 34](#_Toc10185073)

[图3.17 文书推荐更新流程图 34](#_Toc10185074)

[图4.1 xml文书解析代码 40](#_Toc10185075)

[图4.2 文书对象数据 40](#_Toc10185076)

[图4.3 文书类型转换流程图 41](#_Toc10185077)

[图4.4 文书类型转换代码 42](#_Toc10185078)

[图4.5 文书索引结构 43](#_Toc10185079)

[图4.6 文书检索部分代码 45](#_Toc10185080)

[图4.7 文书高级检索界面 45](#_Toc10185081)

[图4.8 文书检索结果列表界面 46](#_Toc10185082)

[图4.9 文书下载主要代码 47](#_Toc10185083)

[图4.10 文书下载界面 48](#_Toc10185084)

[图4.11 文书查看界面 48](#_Toc10185085)

[图4.12 LDA主题模型训练流程图 49](#_Toc10185086)

[图4.13 主题模型训练相关代码 50](#_Toc10185087)

[图4.14 文书主题分布数据图 51](#_Toc10185088)

[图4.15 文书推荐的计算流程图 52](#_Toc10185089)

[图4.16 困惑度趋势图 53](#_Toc10185090)

[图4.17 文书检索模块测试代码 57](#_Toc10185091)

[图4.18 并发数-平均响应时间折线图 58](#_Toc10185092)

# 表目录

[表2.1 Elasticsearch与MySQL概念对比 9](#_Toc10185093)

[表3.1 文书解析用例描述 17](#_Toc10185094)

[表3.2 文书类型转换用例描述 17](#_Toc10185095)

[表3.3 文书索引用例描述 17](#_Toc10185096)

[表3.4 文书更新用例描述 18](#_Toc10185097)

[表3.5 高级检索用例描述 18](#_Toc10185098)

[表3.6 文书排序用例描述 18](#_Toc10185099)

[表3.7 文书下载用例描述 19](#_Toc10185100)

[表3.8 文书收藏用例描述 19](#_Toc10185101)

[表3.9 文书推荐用例描述 19](#_Toc10185102)

[表3.10 文书信息表WsInfo 35](#_Toc10185103)

[表3.11 文书属性表WsAttribute 35](#_Toc10185104)

[表3.12 用户文书表UserWsb 36](#_Toc10185105)

[表3.13 文书主题分布表WsZtfb 36](#_Toc10185106)

[表3.14 文书推荐表WsRecommend 37](#_Toc10185107)

[表4.1 文书处理模块主要类 39](#_Toc10185108)

[表4.2 文书管理模块主要类 43](#_Toc10185109)

[表4.3 文书检索模块主要类 44](#_Toc10185110)

[表4.4 混淆矩阵 54](#_Toc10185111)

[表4.5 LDA和TF-IDF在不同推荐文书下F1值的对比 55](#_Toc10185112)

# 第一章 引言

## 1.1 项目背景

近些年来，人民法院随着互联网的飞速发展提出了信息化的自我革命，贯彻落实创新驱动战略、网络强国战略、大数据战略、“互联网+”行动计划和新一代人工智能发展规划，加快建设智慧法院[马超等，2016]。裁判文书作为法律审判活动记录的载体，完整记录了人民法院的审理过程和诉讼结果，同时也是人民法院如何判定当事人实体权利义务的重要凭证。一份结构完整、要素齐备，逻辑严谨的裁判文书，不仅是当事人享有应有权利以及承担义务的凭证，同时也是上级法院监督下级法院审判活动的重要依据。2013年7月，裁判文书第一次在中国裁判文书网集中公开，截止到2019年3月，已经有6400万篇裁判文书被收录并且公布到中国裁判文书网当中。裁判文书的公开化进一步促进了司法公开公正，对推进中国法治建设具有重大的意义。

基于司法大数据的研究工作相继展开，“人工智能+法律”也成为了研究的热点。将人工智能和文本挖掘应用到法律领域，通过对大量裁判文书数据的有效检索、挖掘、分析以及研究工作，使得法律行业的运行效率更加的高效。随着裁判文书量的不断累积增加，传统的搜索功能仅仅能够满足用户的一些功能性需求。对于一些非功能性需求，例如实时搜索、搜索结果的满意度以及相关文书的个性化推荐等等，都需要进一步的研究开发，以满足新时代下用户的需求，大幅度提高用户满意度。与此同时，随着人工智能的火热，越来越多的研究人员开始把目光聚集在法律领域，比如基于自然语言处理的语义检索；基于机器学习大数据的推荐技术等等。这些前沿科技的研究必然少不了大量文书数据的支撑，通过对有效数据的深入挖掘分析，训练出契合应用场景的算法模型。为了提高有效文书信息的挖掘，帮助科研人员进一步进行相关工作的研究，需要有一款裁判文书检索与推荐系统来解决这个问题。

根据以上论述的背景，开发一款同时具备文书的高效检索以及相关文书推荐的系统对于帮助科研人员工作的推进具有重要的积极影响。该裁判文书检索与推荐系统应该具备以下几个基本功能点：

（1）文书文本解析，文书因其固定的书写格式与规范可以用半结构化的xml文件来表示，通过对xml文书节点的深度遍历，提取出文书重要的属性字段，以便于后续检索的高效实现。

（2）文书类型转换，文书初始的类型格式是xml形式的。为了方便用户下载相关文书时能够直接读取文书的文本信息，系统需要先把xml格式的文书转换成doc格式的文书，只保留文本信息，去除冗余的信息特征。

（3）文书信息检索，系统需要提供用户全文检索的功能，通过文书全文中的任何一个字段都能够检索到相关的文书信息。与此同时，用户还能够通过一些关键信息，比如案由、案号、裁判时间、法院等属性来进行文书的相关检索。

（4）相关文书推荐，系统对所有的文书进行文本预处理从而得到一个完整的语料库，构建一个LDA主题模型，通过语料库的数据信息对模型进行训练。用训练好的模型对每一篇文书进行主题分布的计算。在查看具体一篇文书时，通过计算文书之间主题分布的相似度来对查看文书进行相关的文书推荐。

通过实现以上所述的功能，裁判文书检索与推荐系统能够大幅度提高用户检索文书的速度，在提高效率的同时，更兼顾了检索文书目标的质量，即检索到的结果应当符合用户的检索期望。同时，文书推荐功能可以帮助科研人员建立合适自己应用场景的数据资料，极大地推动科研工作的进展。

## 1.2 国内外研究现状

全文检索第一次出现于20世纪中期，属于情报检索的一种技术。在1959年的时候，世界上第一个全文检索系统由美国匹兹堡大学首先研发成功，即法律情报检索系统。十几年之后，由美国一家科技公司开发的大型全文数据库Lexis正式投入使用，主要为公众提供新闻、法律以及商业经济等相关内容的查询。Lexis系统的成功应用标志着全文检索领域的开端。从20世纪80年代往后，英文全文检索有了长足的发展并且不断地完善，渐渐成为国外信息检索的主流[Kashiji et al., 2004]。

我国的全文检索技术研究是从20世纪80年代开始，相比国外而言起步略微晚了一些，但发展速度十分可观。武汉大学的陈光祚教授首先提倡通过后控词表的技术来改进全文检索的查询效果[陈睿等，1991]。从20世纪80年代中期开始，先后有多个全文数据库投入应用，其中包括人民日报全文数据库以及经济日报全文数据库。从80年代末开始，我国通过对国外全文检索技术的研究，再结合中文汉字的处理特点开始了独立开发全文检索的探索。

进入21世纪，徐建华等著名学者通过对我国过去全文数据库检索进行详细深入的分析，提出了全文检索设计以及开发技术研究的热门问题。与此同时，我国的一些学者也对国外先进的全文检索技术进行了比较深入的剖析[何巨璞，2014]。如《基于WWW的全文检索系统性能探讨》，该文通过比较EBSCOhost和ProQuest Direct这些先进的全文检索系统，深入探讨了基于WWW的检索系统的性能特点[胡琼，2001]。另一方面，一些学者专家基于中文汉字自身的特点，提出了一系列的技术解决方案并不断深入探索构建新的全文检索模式。比如在《全文检索研究》一文中，作者提出了“自动标引算法”，该算法能够避免歧义切分问题并且能够保证查询结果的质量[张卫丰等，2000]。

近些年来，随着互联网技术的快速发展以及互联网用户的暴涨性增长，网络产生的数据呈指数级别的累积增长。针对大数据量的全文检索技术也应运而生，比如Lucene、solr、sphinx以及Elasticsearch等等[马鹏杰，2018]。其中solr和Elasticsearch都是基于Lucene进行的二次开发，使得检索技术更加简洁易于上手[Bai, 2013]。这些先进的技术在这些年也得到了广泛的商业应用。然而在法律行业，很多检索系统还在使用传统的关键字比对技术来进行全文检索。在数据量小的时候差距还不是很明显，但随着数据的不断累积，检索效率也越来越低下。比如近些年，伴随着全国法院信息化的转型，本着依法治国的战略理念，进一步推进司法公开，中国裁判文书网应运而生。但是该系统的文本检索设计思路依然是上世纪九十年代的检索方案，功能也只停留在：对关键字段的匹配查询，或对多个关键字进行完全匹配查询，不支持关联查询以及类似文书的推荐。裁判文书的网上公开表明了我国在司法公开的道路上迈开了坚实而又重要的一步，但从用户体验和当前主流搜索引擎技术看，目前那些稳定并且功能强大的搜索引擎并没有应用到中国裁判文书网。而这样的开源搜索引擎有很多，比如上文提到的Elasticsearch还有分布式搜索引擎[张高伟，2014]。

相比商业项目对于前沿的开源搜索引擎技术的应用，像中国裁判文书网这样的国家公共项目对于新技术的应用显得更加的保守。当然随着用户对系统使用体验要求的不断提高以及开源搜索引擎技术的不断成熟，有必要将先进的检索技术应用到实际项目当中，不断提高系统的运行效率。

## 1.3 本文主要研究工作

本文主要介绍裁判文书检索与推荐系统的两大核心功能检索与推荐，对项目背景以及所需要应用的技术做了相关性的阐述，描述了需求搜集、需求分析、系统设计、功能代码实现以及系统环境部署的全部过程。本文所研究的裁判文书检索与推荐系统是把检索与推荐功能结合起来，帮助用户更高效的检索想要的文书，并且给用户推送可能需要的文书。

裁判文书检索与推荐系统从功能上被划分成文书处理模块、文书管理模块、文书检索模块以及文书推荐模块四个功能模块。其中文书处理模块包括文书解析和文书类型转换，文书解析主要是对xml格式的文书进行读取，遍历整篇文书的所有节点，按照设定好的文书基本特征属性对相应的节点内容进行解析和提取，最后保存到数据库中。文书类型转换是将xml格式的文书转换成word样式的文书。文书检索模块包括对文书索引的建立，通过文书解析得到的文书属性进行Field的建立，方便对文书进行条件检索。在检索过程中，既可以通过一系列条件，比如案由、法院、裁判时间和文书类型等等进行搜索，也可以通过全文检索，即通过文书当中出现的某一个词语进行文书检索。当返回给用户检索结果之后，依然可以通过关键字、文书类型、裁判时间等进行二次结果检索。文书推荐模块是该系统的主要功能之一，主要是通过一篇文书查看其相似的文书。先是构建一个LDA主题模型[Wang et al., 2011]，然后对所有的文书进行读取以及一系列的数据预处理相关工作，再将处理好的数据投入到主题模型中进行计算训练，从而得到一个适用于计算文书主题分布的LDA主题模型[Wang et al., 2017]。紧接着遍历所有的文书，通过训练好的主题模型计算其主题分布，并将结果保存[杨凡，2018]。最后通过计算文书之间主题分布的相似性推算出文本内容之间的相似性，从而给出相关的文书推荐[向李兴，2015]。

裁判文书检索与推荐系统是基于Elasticsearch搜索引擎进行检索[Lei et al., 2015]，不同于传统的关系型数据库的检索，它采用倒排索引的检索方式来提高文书的检索速度[韩冰，2009]。与此同时，该系统采用MVC的设计思想，使得系统具有很强的可扩展性以及可维护性，便于开发人员不断完善已有的功能模块以及拓展新的功能。

## 1.4 本文的组织结构

本文主要是基于裁判文书检索与推荐系统进行展开，并对系统的各个功能模块进行了重点的描述。本文首先介绍裁判文书检索与推荐系统的项目整体背景以及全文检索技术国内外的发展状况，介绍了裁判文书检索与推荐系统在开发过程当中要用到的相关技术。然后对裁判文书检索与推荐系统的功能模块进行了详细的介绍，并重点描述了该系统涉及模块的业务流程、设计以及实现。

第一章为引言部分，主要介绍了系统开发的项目背景以及意义，全文检索技术的发展现状、论文的主要研究工作以及组织结构。

第二章为技术综述，主要介绍了在裁判文书检索与推荐系统开发过程中所使用到的技术，其中包括：MVC模式、Elasticsearch、LDA主题模型、Spring框架。

第三章为裁判文书检索与推荐系统的需求分析与设计，对该系统的需求进行分析，并且介绍了系统的设计思路。其中包括系统中各个功能模块的划分，各个功能模块之间的关系、内部设计和数据库设计。

第四章为裁判文书检索推荐系统的具体实现，在需求和业务流程描述的基础上，详细阐述了文书处理模块、文书管理模块、文书检索模块和文书推荐模块四个模块的实现细节，描述了各个功能模块之间的交互过程。

第五章为总结与展望，对裁判文书检索与推荐系统实现过程以及在撰写论文期间所做工作的总结，提出项目中需要进一步完善的地方以及对未来工作的展望。

# 第二章 技术综述

## 2.1 MVC模式

### 2.1.1 MVC模式的简介

MVC（Model-View-Controller）是一个架构模式，通常用于开发将应用程序划分成三个相互连接部分的用户界面。这样做是为了将信息的内部表示与向用户显示和接收信息的方式分离开来。该模式划分有三个角色：模型、视图和控制器，它将程序中的所有对象分配为这三个角色之一[周嘉程，2018]。这三种角色中每一个对象都通过抽象边界与其他对象分开，并且跨越这些边界与其他类型的对象进行连接通信。该模式包括三个核心组件：

Model（模型）：Model封装的是数据源以及基于对这些数据的所有相关操作。在一个组件当中，Model一般情况下表示组件的状态以及操作状态的相关方法。Model层主要是负责处理由控制器传送过来的各种请求，并将这些请求转换成数据对象状态的更改，而状态的更改则是通过与数据库进行交互，并将更改的状态持久化到数据库当中。与此同时，在Model层的数据和View层的数据没有直接的关联，因此，Model层提供的对象数据能够被多个视图对象所使用，这样能够极大的提高代码的重用性。

View（视图）：View封装的是对数据源Model的一种显示，一个模型可以有多个视图，一个视图也可以由不同的模型对象关联组合起来。View层主要有两个功能职责：将控制层传来的数据通过页面展示给用户查看，将用户的操作请求发送给控制层。换句话说，View层是用户能够看到并且与之交互的界面，是对数据源的一种系统展示。

Controller（控制器）：Controller封装的是用户对模型的各种操作。一般情况下，这些操作最终会转换成对模型的操作，即调用模型中相应的方法。通常Controller在Model和View之间作为一个中介传递信息的存在，处理用户在View上的输入，并将其转发给Model。这样能够降低Model和View两个组件之间的耦合性，使得系统更加易于维护，便于后期应用功能的拓展。

### 2.1.2 MVC模式的优缺点

MVC模式的优点：

（1）一个模型在运行过程中可以同时建立并且使用多个视图模型。变化-传播机制的应用可以确保所有相关联的视图都能够及时根据数据模型的变化而做出实时的更新，使得关联视图和控制器在行为上达到同步效果。

（2）MVC模式因为对应用程序进行分层，每一层都有自己的功能，职责分工明确，最大化实现了分层开发。

（3）各层之间分工明确，极大地降低了层与层之间地依赖关系，便于项目代码地标准化以及后期功能的拓展和维护。

（4）当项目需求发生变更时，只需要修改对应层的代码即可，极大程度地实现了代码的复用。

MVC模式的缺点：

（1）MVC的初衷是为了降低代码之间的耦合，但也不可避免地增加了系统结构地复杂性，使得多出不少冗余地代码。

（2）由于三层结构的设计，系统界面的数据都要经过这三层结构最后到达数据库，使得访问数据库的效率降低了。

## 2.2 Elasticsearch搜索引擎

### 2.2.1 Elasticsearch简介

Lucene是Apache软件基金会开发的一个开源的全文信息搜索引擎框架，提供了完整的查询、索引以及相关文本分析引擎[Taylor et al., 2018]。Lucene是一个快速便捷的基于Java的全文检索库，提供一整套简单而又强大的API，能够实现实时全文检索[宏朴, 2016]。

Elasticsearch是在Lucene的基础之上进行开发的一款开源搜索引擎。由于Lucene只是一个搜索引擎库，如果想要在程序开发过程中使用它的话，必须使用开发语言Java将其集成到开发应用当中[曾亚飞，2016]。但是由于Lucene的复杂性，开发人员需要对其有较为深入的理解才能更好的应用。Elasticsearch是用Java语言进行开发，并通过Lucene作为其核心组件来实现索引和搜索功能，同时又通过RESTful API来封装Lucene的功能，让使用更加的简单便捷[肖运文，2016]。Elasticsearch具有如下特征：分布式实时文件存储，索引并且搜索每个字段；分布式的实时搜索引擎；可以同时部署到多个服务器上，处理非结构化或结构化数据[钦蒋承等，2018]。

### 2.2.2 Elasticsearch基本概念

索引（Index）：Elasticsearch把数据存储在一个或者多个索引当中，它是具有相似特征的文档的集合。相比传统的关系型数据库而言，索引类似于SQL当中的一个数据库，又或者是一个数据存储方案（schema）。

类型（Type）:类型是索引库中的一个逻辑分区，用户可以自己根据项目的需要来进行定义。因此，一个索引库中可以定义有一个或者多个类型（Type）。

文档（Document）： 文档是包含了一个或者多个域的容器，是用JSON格式来表示的。文档由一个或者多个域组成，每一个域有一个名字以及一个或者多个值，拥有多个值的被称为“多值域”。每个文档能够存储不同的域集，但是在一个类型下的文档应该具有很高的相似性。

集群（Cluster）：Elasticsearch集群是多个节点的集合，它们一起存储所有的数据集，同时提供联合索引和跨节点索引的能力。多节点构成的集群会有一定的冗余，因此当一个或者几个节点出现问题时，会由它的备用节点替代，不会出现单点故障，保证了服务的整体可用性。

节点（Node）：节点是运行单个实例的Elasticsearch主机，它是集群中的成员，能够进行存储数据、参与集群的相关操作。

分片（Shard）和副本（Replica）：Elasticsearch “分片”机制将索引中的数据存储在多个节点当中，通过把一个索引分成多个底层的Lucene索引来完成索引数据的分割和存储功能，其中每一个物理索引被称为一个分片（Shard）。每一个分片的内部都是一个功能齐全并且独立的索引，所以能够让集群中的任何一个主机来存储。在创建索引时，用户能够指定分片数量。

为了便于理解各个基本的概念，将Elasticsearch和传统的关系型数据库（MySQL）进行对比，内容如表2.1所示：

表2.1 Elasticsearch与MySQL概念对比

|  |  |
| --- | --- |
| Elasticsearch | MySQL |
| Index | Database |
| Type | Table |
| Document | Row |
| Field | Column |
| Mapping | Schema |
| Everything is indexed | Index |
| Query DSL | SQL |
| GET http://.. | SELECT \* FROM table |
| PUT http://.. | UPDATE table SET… |

### 2.2.3 Elasticsearch架构

Elasticsearch主要是提供分布式的搜索框架用来拓展Lucene，以此来实现大数据条件下的分布式搜索的需求，从而达到快速安装部署，实时搜索，方便使用的效果。其核心实现思想比较简单，就是分而治之，把大的数据量分成了一个一个小的分片。当要执行任务时，对每一个分片进行处理，将每一份结果一一汇总合并，最终返回给用户。Elasticsearch的体系架构如图2.1所示：



图2.1 Elasticsearch体系架构

（1）存储层GateWay

GateWay代表了索引快照的存储方式，Elasticsearch是优先将索引存储在内存当中，在内存满了之后再存储到本地硬盘。GateWay对索引快照进行存储，当集群关闭后再重新启动时会从GateWay中读取备份数据。

（2）实现层Distributed Lucene Directory

该层是Lucene的分布式框架，Lucene是做索引的，但它是一个单机的搜索引擎，要实现Elasticsearch这种分布式搜索引擎系统，需要在每个节点上都能够通过Lucene进行索引、查询以及更新等相关操作。

（3）功能层 Function layer

该层主要有：数据源、索引管理、检索等功能模块。数据源是Elasticsearch同步数据源的一种方法，主要通过插件将数据源读取并索引到Elasticsearch中；索引管理主要包括索引的创建，修改以及删除；检索主要是根据搜索条件进行查询，并且返回查询得到的结果。

（4）接口层 RESTful style API

这是Elasticsearch提供的访问接口，可以直接发送HTTP请求，方便后续使用nginx做代理，分发包括后续可能会做的权限管理。

## 2.3 LDA主题模型

### 2.3.1 LDA主题模型简介

LDA（Latent　Dirichlet　Allocation）中文意思是：潜在狄利克雷分布[Blei et al., 2003]。LDA主题模型是一种文档生成模型，属于无监督的机器学习技术。它认为一篇文档是可以有多个主题的，并且每一个主题都有它的词分布[Rosen-Zvi et al., 2004] 。它假定一篇文档的构造首先是从一定的概率选择某一个主题，然后在这个主题下再以一定的概率选择某一个词，这样就产生了文档的第一个词。紧接着，不断重复这个过程，一直到最后就生成了整篇文档[徐戈等，2011]。

上面描述的是一篇文档的生成过程。LDA的使用是文档生成过程的逆过程，也就是通过一篇文章去推测这篇文档对应的主题，再通过主题来计算相应的词。

### 2.3.2 LDA主题模型的原理

图2.2是LDA主题模型的概率图表示：有阴影部分的圆圈代表着能够被观察到的变量；白色的圆圈代表着潜在的变量；方框则是表示里面变量的集合。



图2.2 LDA概率图

模型的参数描述：

方框中：K表示主题的数量，N表示一篇文档中词的数量，M表示要分析的文档的数量。α表示每篇文档主题分布的先验分布（即Dirichlet分布）的参数，β同样表示每个主题词分布的先验分布（即Dirichlet分布）的参数，表示主题k下的词分布，表示文档i下的主题分布，是的主题分配，表示第i篇文档中的第j个词。

所以，对于文档d中的每一个单词，LDA会根据先验知识α确定这篇文档的主题分布θ，紧接着从该篇文档所对应的主题分布（多项分布）θ中按照一定的概率抽取一个主题z，然后再根据先验知识β确定当前主题z下的词分布φ，接着从主题z所对应的词分布（多项分布）φ中以一定的概率抽取一个单词w。后面将这个过程循环反复N次，最后就产生了文档d。

从另一个角度来说：

（1）假定语料库中一共有M篇文档，每篇文档下的Topic的主题分布是从一个参数为α的Dirichlet先验分布中抽取得到的多项分布。每一个Topic下的词分布是从一个参数为β的Dirichlet先验分布中抽取得到的多项分布。

（2）对于一篇文档中的第n个词，第一步是从该文章的每一个主题分布中采样或者选择一个主题，第二步则是在这个主题相应的词分布中采样或者选择一个词。不断地循环反复这个随机生成过程，一直到M篇文档全部都生成完毕。

以上讲述的是LDA模型中文档的生成方式以及流程，可以发现里面有一部分可见的变量以及一部分不可见的变量，要通过对未知参数的求解进一步得到完整的LDA模型。所有可见变量和隐藏变量的联合分布如公式（2.1）所示：

公式（2.1）

最后一篇文档的词分布的最大似然估计能够通过将上面式子中的和Φ进行积分以及对进行求和得到，计算方式如公式（2.2）所示：

公式（2.2）

根据的最大似然估计，最后能够通过Gibbs采样等方法估计出模型中的参数。

## 2.4 Spring框架

### 2.4.1 Spring介绍

Spring是一个开源的Java框架，目的是解决复杂的企业级应用程序开发的问题[陈雄华等，2012]。最初为了解决这个问题，创建了J2EE（Java 2 Platform Enterprise Edition）。但是J2EE需要实现不同类型的各种接口，配置复杂而又单调，开发效率较低。Spring的出现就是为了实现J2EE未能完成的目标，致力于提供一套完整的，统一的，高效的方式来构造整个项目应用，并且能够把单层架构以最优的方式组合成一个连贯的体系[黄睿，2007]。简而言之，Spring就是一个帮助开发人员进行J2EE开发的框架。

Spring是一个分层架构，有7个模块。核心容器是Spring框架的基础，通过依赖注入来实现容器对Bean的管理。控制反转模块通过IOC技术来实现松耦合的效果。当运用IOC时，一个对象所依赖的其他对象会被动地传递进来，而不是自己去查找或创建依赖对象[蒲奇等，2008]。IOC是一种设计模式，通过Spring框架来实现对象的创建和管理，用户只需要关注对象的创建，而不需要关注对象之间的依赖关系。面向切面（AOP）模块对面向切面编程提供丰富的支持，它将应用的业务逻辑和系统服务（权限管理、系统日志等）分离开来进行内聚性开发[袁绪峰，2006]。JDBC抽象和DAO模块的操作流程包括取得连接、创建语句、处理数据集以及关闭连接等操作，这中间会产生大量重复的代码。Spring抽取重复的代码，使得访问数据库的代码更加简洁，同时还可以防止关闭数据库失败引起的异常问题。对象/关系映射集成模块当中，Spring不需要自己实现ORM解决方案，而是给流行的几种ORM框架提供集成方案。其中包括JDO、Hibernate以及iBATIS SQL映射。Web模块当中，Spring在应用上下文模块之间建立Web上下文模块，同时给这个模块提供面向服务支持。

Spring框架因其控制反转以及面向切面的特点，能够帮助开发人员在开发项目时减少许多复杂冗余的工作，只需要专注业务逻辑的开发即可。Spring框架模块图如图2.3所示：



图2.3 Spring框架图

### 2.4.2 Spring MVC介绍

Spring MVC是基于Java的轻量级Web框架，它实现了MVC设计模式下的请求驱动。也就是用MVC架构思想对Web层进行职责解耦[周燕玲，2016]。其中请求驱动是指请求－响应模型，帮助开发人员简化开发工作[薛峰等, 2012]。Spring MVC有着清晰的角色划分，每一个角色都是由专门的对象来实现，使得项目的结构更加清晰，大幅度提高开发效率。其工作流程为：

（1）第一步用户发送请求给Dispatcher Servlet（前端控制器），前端控制器接收请求后委托其他解析器进行处理，它作为统一访问点进行全局流程控制。

（2）前端控制器请求Handler Mapping（处理映射器）查找Hander，可根据XML配置、注解进行查找。

（3）处理映射器向前端控制器返回Handler，前端控制器调用处理器适配器去执行Handler。

（4）Handler执行完成后给适配器返回ModelAndView（Spring MVC的一个底层对象，包括Model和View），处理适配器再向前端控制器返回ModelAndView。

（5）前端控制器请求视图解析器进行视图解析，视图解析器将解析出的view返回给前端控制器

（6）前端控制器进行视图渲染，即将模型数据填充到request域中并向用户响应结果。

Spring MVC工作流程图如图2.4所示：



图2.4 Spring MVC工作流程图

## 2.5 本章小结

本章主要介绍了裁判文书检索与推荐系统中使用的相关主要技术，并且阐述了使用该技术的原因。包括有：MVC模式，Elasticsearch搜索引擎，LDA主题模型以及Spring框架。其中，由于MVC模式简洁明了，易于维护，易于拓展的特性，选择其作为裁判文书检索与推荐系统的设计模式。

# 第三章 裁判文书检索与推荐系统的分析与设计

本章节主要是介绍裁判文书检索与推荐系统的设计。首先通过用例图和用例描述阐述系统的功能性需求和非功能性需求。紧接着通过各类视图对系统总体设计进行描述，并给出每一个模块的详细设计，最后简要地介绍系统的数据库设计。

## 3.1 系统概述

裁判文书检索与推荐系统顾名思义是集检索与推荐于一体的系统项目，目的是帮助用户更加高效便捷地检索想要的文书。用户使用系统可以进行高级检索，并对检索到的文书列表进行相关性排序，比如法院层级、裁判日期和审判程序。同时，系统提供用户批量文书下载以及文书收藏的功能，提高用户的检索体验。当用户查看具体某一篇文书时，系统会根据查看的文书默认推荐相似性较高的文书供用户查看，用户也可以选择查看更多相似性高的文书。

裁判文书的原始数据形式是xml格式的，系统需要对xml文书进行解析从而提取出文书的基本属性信息。与此同时，为了方便用户能够多类型下载文书，系统需要对文书类型进行转换，这样能够保证用户下载不同类型文书的效率。用户能够进行文书检索的前提是已经对文书创建了索引，该索引是系统通过解析的文书对象来进行创建。系统所包含的文书集是动态变化的，需要对新增的文书集进行更新，包括文书解析、文书类型转换、文书索引创建以及文书推荐更新等一系列相关操作。这些系统操作是对文书数据的前置处理，是用户使用系统进行检索和推荐操作的基础。

## 3.2 系统需求分析

### 3.2.1 系统用例

根据需求分析，裁判文书检索与推荐系统主要有系统用户以及系统两种角色。用户能够使用高级检索即按照条件进行文书检索，比如：案由、案件名称、案号、案件类型、审判程序以及文书类型等等。当系统根据检索条件返回结果给用户，用户可以根据一定条件对文书结果进行排序，同时也可以对文书进行批量下载以及收藏操作。对于系统角色而言，需要对最原始的xml文书进行遍历解析，提取出节点中文书的基本属性信息，并且通过提取的基本属性信息进行索引创建。同时，为了方便用户doc文书的下载，系统需要提前将xml格式的文书转换成doc格式。由于文书集是动态变化的，系统需要对变动的文书集进行更新。裁判文书检索与推荐系统用例图如图3.1所示：



图3.1 系统用例图

### 3.2.2 功能需求

裁判文书有标准的书写格式，包括有文首、当事人、诉讼记录、案件基本情况、裁判分析过程、判决结果以及文尾七个组织结构，每一个组织结构也都有其固定的书写规范。因此，裁判文书可以用半结构化的xml语言来进行描述。系统需要通过解析xml文书提取出重要的信息特征来对文书进行索引的创建，方便后续用户的检索。同时，还需要将xml格式的文书转换成doc格式的文书，方便后续用户的下载。当出现新的文书集时，系统需要对新增文书进行更新。用户的功能包括有：高级检索、文书排序、文书下载、文书收藏以及文书推荐。以上用户的功能需求都是建立在系统完成前置工作的基础上才能够完整运行的。

根据图3.1的系统用例图以及相关文字描述，具体分析了裁判文书检索与推荐系统的功能需求，各个功能用例描述如表3.1-表3.9所示。其中，表3.1-表3.4是关于系统的用例描述，表3.5-表3.9是关于用户功能需求的用例描述。各个功能的用例描述如下：

表3.1 文书解析用例描述

|  |  |
| --- | --- |
| ID | UC1 |
| 名称 | 文书解析 |
| 详细描述 | 系统读取xml文书，逐一遍历xml节点信息，根据文书属性表提取相应的信息。由于案件类型的不同，一些节点的标注也不相同。所以在遍历的过程中需要判定案件类型，包括有民事案件、刑事案件、行政案件、赔偿案件以及执行案件。通过不同的案件类型获取不同的节点信息，最终得到文书的基本属性信息。最后对创建好的文书基本信息对象进行保存。 |
| 优先级 | 高 |
| 数据输入 | xml文书 |
| 数据输出 | 返回文书的基本信息对象 |

表3.2 文书类型转换用例描述

|  |  |
| --- | --- |
| ID | UC2 |
| 名称 | 文书类型转换 |
| 详细描述 | xml文书根节点是全文节点包括了所有的文书信息，其又包含了七个子节点，分别是文首、当事人、诉讼记录、案件基本情况、裁判分析过程、裁判结果以及文尾。其中，每一个节点都包括有完整的文字信息。将每一个节点的信息提取出来，根据提前准备好的word文书格式进行信息的填充，最后形成一篇和xml文书名称相同的doc文书。 |
| 优先级 | 高 |
| 数据输入 | xml文书 |
| 数据输出 | doc文书 |

表3.3 文书索引用例描述

|  |  |
| --- | --- |
| ID | UC3 |
| 名称 | 文书索引 |
| 详细描述 | 对于解析好的文书对象，系统需要对其进行索引创建。首先，创建新的索引并对索引进行配置设置，比如索引的主分片以及每个主分片的副本数。然后再创建这个索引的mapping，其中可以定义mapping的数据类型以及是否需要分词。接着将解析好的文书对象依次插入到索引，便于后续文书检索。 |
| 优先级 | 高 |
| 数据输入 | 文书对象 |
| 数据输出 | 文书索引 |

表3.4 文书更新用例描述

|  |  |
| --- | --- |
| ID | UC4 |
| 名称 | 文书更新 |
| 详细描述 | 系统的文书集是不断增加的，对于新增的文书集，系统需要及时进行更新操作。首先遍历所有的新增文书，调用文书解析功能对xml文书进行解析，然后将解析好的文书对象插入到索引当中，并对所有新增文书进行类型转换。在后续的推荐功能中，需要通过算法模型计算新增文书的主题分布，并且遍历所有的文书时，计算文书之间的主题分布分歧度。增加新文书的推荐文书，并对原有文书的推荐文书进行更新。通过对新增文书的及时更新，保证系统后续检索与推荐效果的及时性和准确性。 |
| 优先级 | 高 |
| 数据输入 | 新增文书 |
| 数据输出 | 更新好的索引和推荐表 |

表3.5 高级检索用例描述

|  |  |
| --- | --- |
| ID | UC5 |
| 名称 | 高级检索 |
| 详细描述 | 高级检索分为按条件检索和按全文检索。由于之前文书解析过程中已经提取出文书的基本信息，包括有案由、法院层级、案件类型、年份、审判程序、文书类型等筛选信息，用户可以在下拉框中选择。还有一些信息比如案件名称、案号、法院名称、法律依据等输入信息，用户可以自行进行输入。当用户点击检索按钮，系统会根据用户的查询条件查找符合用户条件的文书，并将结果返回给用户查看。同时，用户依旧可以在检索结果的基础上进行二次条件检索。用户可以选择关键词、案由、法院层级、地域及法院、裁判年份、审判程序、文书类型等条件继续进行检索。同时，用户可以对检索条件进行单个删除，提高检索方式的多样化。 |
| 优先级 | 高 |
| 数据输入 | 用户输入查询条件 |
| 数据输出 | 系统返回检索文书结果 |

表3.6 文书排序用例描述

|  |  |
| --- | --- |
| ID | UC6 |
| 名称 | 文书排序 |
| 详细描述 | 系统返回用户检索得到的文书列表，系统默认按照法院层级的递减顺序进行排序。系统提供三种排序方法，分别是法院层级、裁判日期、审判程序，用户可以选择排序类型，并选择排序方式是递增还是递减。系统会对检索的文书集进行相关排序并将排序后的文书结果返回给用户。同时系统会提示用户当前的排序条件以及排序方式。 |
| 优先级 | 高 |
| 数据输入 | 排序条件 |
| 数据输出 | 排序后的文书列表 |

表3.7 文书下载用例描述

|  |  |
| --- | --- |
| ID | UC7 |
| 名称 | 文书下载 |
| 详细描述 | 文书下载包括xml文书下载和doc文书下载。用户在检索之后，系统会返回文书列表。用户可以选择一篇文书下载，也可以选择多篇文书下载。当下载多篇文书时，即批量下载，系统会将所有要下载的文书打包，以下载时间重新命名。如果只有一篇文书需要下载，无论是xml格式还是doc格式，文书名称和原来保持一致。在推荐文书列表中，用户依然可以选择单篇下载或者多篇下载，也可以选择不同文书类型进行下载。在查看具体一篇文书时，用户可以直接选择下载单篇文书。 |
| 优先级 | 高 |
| 数据输入 | 文书列表 |
| 数据输出 | 下载文书结果集 |

表3.8 文书收藏用例描述

|  |  |
| --- | --- |
| ID | UC8 |
| 名称 | 文书收藏 |
| 详细描述 | 用户根据检索条件查询到相关文书，当用户对某一篇文书感兴趣时，可以选择收藏该篇文书，便于后续的查看。当用户点击文书的收藏按钮时，系统会在浏览器cookie中生成一个唯一id并将其与文书进行关系映射，用户在查看收藏列表时也可以选择取消文书收藏。 |
| 优先级 | 高 |
| 数据输入 | 文书列表 |
| 数据输出 | 无 |

表3.9 文书推荐用例描述

|  |  |
| --- | --- |
| ID | UC9 |
| 名称 | 文书推荐 |
| 详细描述 | 在用户查看具体一篇文书时，系统会默认推荐相似度很高的前十篇文书以供用户点击查阅。同时系统也会提供推荐更多相似文书的功能，这时候系统会推荐大概上百篇的文书集，这时候系统会用列表界面的形式将推荐结果呈现给用户。当用户点击其中一篇文书进行查看时，系统会根据此时用户查看的文书进行相似度推荐，从而进行了一轮循环。 |
| 优先级 | 高 |
| 数据输入 | 裁判文书 |
| 数据输出 | 推荐文书结果集 |

### 3.2.3 非功能需求

（1）可用性(usability)

系统能够实现功能需求中所要求的基本操作，功能可用并且完整。同时，用户能够快速上手使用系统，操作简单便捷，学习成本较低。这要求系统有着简单清晰的流程控制。

（2）可靠性(reliability)

保证系统在运行过程中，不会无故发生故障。当用户进行不规范的操作时，系统不会崩溃停止运行，而是提示用户正常的系统操作。也就是系统应该有良好的异常处理机制。

（3）性能(performance)

用户进行文书检索时，将检索条件传到后台搜索引擎进行计算，然后将检索到的文书展现给用户查看，这一过程消耗时间尽量保证在1秒以内。用户查看具体一篇文书，当点击更多文书推荐时，系统根据用户查看的文书进行相似度计算，然后将推荐结果返回给用户，这一过程消耗时间保证在1秒以内。

## 3.3 系统总体设计

### 3.3.1 系统模块划分

裁判文书检索与推荐系统根据系统需求分析按照功能划分，主要由文书处理模块、文书管理模块、文书检索模块以及文书推荐模块组成。系统功能模块如图3.2所示，详细描述如下：

（1）文书处理模块

文书处理模块主要包含文书解析和文书类型转换两个功能点。文书解析功能是指系统需要读取xml文书并对其进行解析，提取出文书相关基本属性信息并保存到文书对象中。文书类型转换功能是指为了方便后续用户对各种类型的文书进行下载，系统需要事先将xml文书转换成对应的doc文书，同时将生成doc文书的信息保存到文书对象中，方便下载时定位到该文书。

（2）文书管理模块

文书管理模块主要包含文书索引和文书更新两个功能点。文书索引是对解析好的文书对象构建索引，方便后续高效检索。系统需要创建新的索引，并创建该索引的mapping用来存储文档域，最后将文书对象保存到索引当中。文书更新是对新增文书的操作，其中包括了文书处理模块和文书推荐模块的操作。

（3）文书检索模块

文书检索模块主要包括高级检索、文书排序、文书下载和文书收藏等功能。用户根据系统检索提示输入相关检索条件，系统返回给用户相关的检索文书列表。用户除了可以点击具体一篇文书进行查看，也可以对检索的文书结果进行相关性排序，比如法院层级、裁判日期、审判程序。用户还可以对文书集进行批量下载，同时选择下载文书的类型。对于感兴趣的文书，用户可以进行收藏，当用户点击我的收藏按钮时，可以查看收藏的文书列表。

（4）文书推荐模块

文书推荐模块主要包含主题模型训练和文书推荐两个功能。系统需要对文书进行预处理操作，包括文本分词、去除停用词、训练词向量等等。然后设定LDA模型主题数并进行训练，通过训练好的主题模型计算文书的主题分布，再根据主题分布计算分歧度，得到相似性较高的文书并进行推荐。同时，为了确保推荐质量，需要对主题模型以及推荐效果进行评估。



图3.2 系统功能模块图

### 3.3.2 系统逻辑视图

裁判文书检索与推荐系统逻辑上分为视图层、控制层、服务层和数据层。视图层提供页面让用户能够输入检索条件，其中包括文书的基本属性条件以及全文检索字段。控制层处理视图层页面传送的数据，根据用户操作调用对应的服务层。服务层提供业务逻辑处理，数据层负责数据的持久化。视图层由JavaScript、HTML、CSS等组件构成，视图层需要接收发送的数据通过Http发送异步请求与控制层连接，控制层由不同业务的Controller组件构成，服务层由不同业务的Service组件构成，数据层由不同业务的Dao组件构成。具体如图3.3所示：



图3.3 系统逻辑视图

### 3.3.3 系统开发视图

裁判文书检索与推荐系统主要是由文书处理模块、文书管理模块、文书检索模块以及文书推荐模块组成。整个系统的所有功能模块都是建立在Elasticsearch集群的基础上，文书处理模块是整个系统的起始功能模块，它是为后面的功能模块提供数据支持服务。文书管理模块是对解析的文书对象数据进行索引创建以及新增文书的更新。文书检索模块对用户提供检索相关的一系列服务操作，包括高级检索、文书排序、文书下载以及文书收藏。文书推荐模块则是通过文书数据来训练LDA主题模型进而通过算法模型来计算主题分布并计算出相似度较高的文书。文书处理模块和文书管理模块是基础模块，是为检索和推荐模块服务。系统开发视图如图3.4所示：



图3.4 系统开发视图

### 3.3.4 系统物理视图

裁判文书检索与推荐系统的物理视图如图3.5所示，主要分为客户端、服务端和数据端三个部分。客户端组件是系统用户PC上的浏览器，比如火狐、Chrome、IE等等。服务端组件部署在服务器主机，该主机提供Tomcat、JDK1.8等系统运行环境。与此同时，Elasticsearch搜索引擎也要部署在该服务器上，由于Elasticsearch是分布式的搜索方式，可以在服务器上搭建一个数据集群，部署一个主节点以及多个副节点。



图3.5 系统物理视图

### 3.3.5 总体结构

系统的展示界面也就是表示层，使用Web浏览器来查看系统运行效果，由HTML来描绘元素，CSS来控制页面样式。前端采用的是Bootstrap框架，使用Thymeleaf模板来渲染界面，并且通过AJAX异步交互负责前后台数据的传输。服务层是以功能模块来进行划分，包括有文书处理模块、文书管理模块、文书检索模块以及文书推荐模块。各个模块都存在业务逻辑的处理以及数据的交互，这些都是通过Spring框架来进行统一管理调度。在数据层，使用的是Elasticsearch集群，所有的文书相关信息都存放在集群当中。为了集群的高效计算会设置多个节点，每个节点会有多个分片。集群中会保存文书解析得到的文书基本属性表，文书信息表，通过LDA主题模型计算得出主题分布表以及文书推荐表。为了系统的完整运行，要保证数据库中文书的ID要是全局唯一的，各个表同一篇的文书应该是同一个ID，这样能提高系统的检索速度，不会出现检索错误。与此同时，xml文书以及文书类型转换得到的doc文书需要存储在本地服务器上，当用户需要进行下载操作时，可以直接从服务器上根据文书基本信息进行查找并对文书进行下载。裁判文书检索与推荐系统的分层架构图如图3.6所示：



图3.6 系统架构图

## 3.4 系统模块设计

### 3.4.1 文书处理模块

文书处理模块主要包含文书解析和文书类型转换两个功能点，由于系统接收的源数据是xml格式的文书，需要对xml文书进行解析并提取相关属性信息。同时为了支持系统能够进行不同类型的文书下载，需要系统在后台对文书进行类型转换操作，下面分别对两个功能设计进行描述。

（1）文书解析功能

文书解析功能主要是负责读取并解析xml文书，提取出文书的关键基本属性信息。该系统使用的是Dom4j的方式来解析xml文书，首先需要创建一个解析器reader，指定需要解析的xml文书。使用解析器来读取xml文书，获取根元素的信息，得到节点名称以及文本值。判断根元素是否存在子元素，如果不存在就结束解析过程，存在的话遍历读取根元素的子元素。设置一个读取元素的函数，该函数能够读取元素节点名称以及文本值。递归调用自身方法，判断该元素是否还存在子元素，以此类推并获得信息。在遍历节点的过程中，需要判断文书的案件类型，比如民事案件、刑事案件以及行政案件等等。不同的案件类型文书xml的节点名称也有细微的差别，同时提取出与文书属性表中对应的节点文本值。后续文书更新时，系统需要重新调用文书解析功能，对新增文书进行解析操作。文书解析的流程图如图3.7所示：



图3.7 文书解析流程图

（2）文书类型转换功能

文书类型转换的原始数据是文书信息。在文书类型转换之前，首先需要制作一个含有freemarker标签的word文档，该文档定义了需要展示的文书内容以及格式，其中文书内容通过标签${variableName}来进行表示，当制作好word文档后将其另存为Word 2003 XML文档（\*.xml）格式的模板文件。文书内容是存储在数据集群中，用定义好的Java对象进行实例化，并通过之前定义的标签将数据一一写入文件，然后将xml模板生成word文档，这时它的本质还是xml，点击另存为依然是xml格式。最后通过调用转换docx方法，将xml格式的doc转换为docx格式的word文档。在转换的过程中，要保证往标签中填入的数据是存在的，如果数据不存在，要默认其跳过，不然会使文书类型转换报错。最后，要将转化得到的doc文书保存到本地服务器上。与此同时，doc文书的文件名称要和其xml文书的文件名称保持一致，保证文书信息的统一。当文书数据更新后，系统要重新调用文书类型转换功能，保证文书信息的及时性。文书类型转换功能的数据流如图3.8所示：



图3.8 文书类型转换数据流图

### 3.4.2 文书管理模块

文书管理模块主要包括文书索引和文书更新两个功能点。文书索引主要是对解析的文书对象创建索引并保存到索引库当中，文书更新负责对新增的文书进行相关功能操作。

（1）文书索引

文书索引功能主要是对要检索的文书创建索引。由于该系统的检索是使用Elasticsearch搜索引擎来实现，文书的索引需要保存在Elasticsearch集群当中。首先，系统需要创建一个index，并设置主分片以及副分片个数。然后创建索引的mapping，设置相关属性，比如是否进行分词。系统需要先获取文书解析好的文书对象，并对文书对象的属性进行处理使得能够与mapping属性一一对应。在索引过程中，系统会对设置分词的域进行分词处理，并全文构建倒排索引，使得能够通过字段能够直接索引到相关文档。最后将建立好的索引保存到索引库当中。当系统检测到新增文书后，系统通过文书解析把新增的文书对象保存在索引库当中。文书索引过程如图3.9所示：



图3.9 文书索引过程图

（2）文书更新

文书更新功能主要是新增文书进行更新操作，需要调用文书处理模块中的文书解析和文书类型转换以及文书索引等功能。一方面需要将新的文书进行解析并对其进行索引创建，同时还需要对新的文书进行类型转换。在文书推荐模块中，也需要对文书进行更新操作，这一部分设计会在推荐模块进行阐述。文书更新流程如图3.10所示：



图3.10 文书更新流程

### 3.4.3 文书检索模块

文书检索模块是该系统的重要核心功能之一。该模块主要包括高级检索、文书排序、文书下载以及文书收藏等功能点。

（1）高级检索

系统前端页面提供高级条件检索，包括案由、案件名称、案号、法院名称、法院层级、案件类型、审判程序、文书类型、裁判日期、审判人员、当事人、律所、律师、法律依据以及年份。用户可以不需要填写全部的检索条件，只需要填写一部分条件即可进行条件检索。

用户在系统界面输入检索条件信息，JS获取表单中的检索并将其通过AJAX的方式传送到后台。后台SearchController类是负责接收前台传入的方法并进行相关处理，再调用ESService类的getWsInfoList方法来对文书进行检索。首先需要调用SearchClientProvider类连接Elasticsearch客户端，通过调用查询方法来对文书进行检索。ESService类的getWsInfoList方法得到文书检索结果，将处理好的文书返回给SearchController类，该类再将检索的文书结果返回到前台。高级检索的过程如图3.11所示：



图3.11 高级检索序列图

由于系统采用MVC设计模式，业务数据需要经过一层一层的处理将数据传输到Elasticsearch搜索引擎进行处理。用户界面的检索条件由searchVo进行封装通过AJAX传到后台SearchController控制类，该类对searchVo进行解析，并分别加入到精确查询和模糊查询中。然后，调用ESService接口类中的方法，该接口由ESServiceImpl类来实现其方法，也就是该类负责条件检索的业务逻辑处理。由于检索基于Elasticsearch，它的操作是通过客户端来实现的，因此调用SearchClientProvider类，该类负责Elasticsearch客户端的创建和关闭。最后通过ESServiceImpl类对搜索引擎进行相关操作，并将检索的结果返回给SearchController类，继而返回到用户界面。该检索的类图如图3.12所示：



图3.12 高级检索类图

（2）文书排序

文书排序功能是用户检索到文书结果集对其进行相关性排序。当用户查看检索到的文书集时，用户可以选择法院层级、裁判日期以及审判程序等条件来进行递增或者递减排序。系统返回给用户查看的文书列表默认是以法院层级来进行排序。由于法院层级、裁判日期和审判程序都是文书对象的属性，并且有对应的索引域，只需在检索时加入排序条件就能实现文书的相关性排序。

（3）文书下载

文书的下载包括xml文书的下载以及doc文书的下载，用户在浏览文书列表或者具体文书时可以选择下载doc或者xml格式的文书，当用户请求通过AJAX方式传送到后台DownloadController类，该类发送url请求给服务器，服务器根据请求解析url并将文件所在的路径返回给DownloadController类。这时新建一个输入流，根据返回的文书路径定位到文书，并将文书返回。再新建一个输出流，将文书下载到本地，并返回一个结果给客户端。当然，有多种下载方式，比如有单篇文书的xml格式下载、单篇文书的doc格式下载、多篇文书的xml格式下载以及多篇文书的doc格式下载。DownloadController根据请求类型对结果进行处理，如果是多篇文书集，就将其压缩成包，压缩包名称为下载时间日期以及下载文书篇数。如果单篇文书，就将单篇文书下载，文书名称保持不变。文书下载的序列图如图3.13所示：



图3.13 文书下载序列图

（4）文书收藏

文书收藏功能是指用户查看相关文书时对感兴趣的文书进行收藏，便于以后直接查看。当用户点击文书收藏时，系统前台会发送http请求，该请求包含浏览器的cookie信息，判断cookie中是否含有用户ID。如果没有，说明是新用户又或者没有进行过文书收藏操作，系统生成唯一的用户ID并保存到cookie中，然后将用户ID和文书进行绑定。如果有用户ID，直接进行绑定。当用户查看收藏文书时，根据用户ID获取所有绑定的文书。文书收藏的流程图如图3.14所示：



图3.14 文书收藏流程图

### 3.4.4 文书推荐模块

文书推荐模块是裁判文书检索与推荐系统的另外一个重要核心功能。文书推荐主要包括三个阶段，一是LDA主题模型构建阶段，二是使用主题模型计算文书主题分布阶段，三是利用主题分布进行文书推荐阶段。其中主题模型构建阶段主要包括三个步骤，分别是：

（1）从数据库中读取所有的裁判文书集，构建训练语料。由于训练的模型需要涉及所有的文书数据集，这样才能够保证模型能够计算所有文书的主题分布。

（2）文书预处理，包括抽取文书的文本信息，并对其进行中文分词，通过专有的裁判文书停用词库对分词后的文书去停用词。这样能够排除文书中对训练模型结果没有作用甚至可能会是干扰点的词语。

（3）训练主题模型，先构建好初始状态的LDA模型，设定主题数，然后将训练语料投入到模型中进行训练，从而得到一个能够计算所有文书主题分布的算法模型。

第二阶段遍历所有的文书，用训练好的主题模型对每一篇文书进行计算得到文书的主题分布，并将结果以键值对的形式保存到数据库中，以便后续推荐的读取使用。

第三阶段是利用主题分布计算文书之间的相似度，从而给出推荐文书。也就是当需要查看具体一篇文书的推荐文书时，遍历计算其他文书与这篇文书的主题分布相似度，然后根据相似值排序得到文书的推荐列表。整个文书推荐的框架总览图如图3.15所示：



图3.15 文书推荐框架总览图

文书推荐是基于文书之间主题分布相似度的计算得到，但是在实际工程实践中，如果要推荐具体一篇文书的其他相似文书，需要读取文书主题分布再计算所有文书与该文书的主题分布相似度，每次都需要遍历所有的文书进行计算并排序，这样工作量太大，会很大程度的影响文书的推荐效率。考虑到文本内容的基本不变性，可以事先计算好每篇文书的相似性文书，由高到低进行排序，并将其保存到数据库中。考虑到数据量的大小，只选择相似度高的top20进行存储，这样在推荐某一篇文书的相似文书时，直接可以从表中读取推荐的20篇文书。当然，由于需求的变化，用户可能不仅需要推荐20篇文书，还可能需要推荐几十甚至上百篇文书。采用广度遍历的算法思想，考虑到目标文书最相似的20篇文书，它们最相似的20篇文书也与目标文书相似，从而对这400篇文书进行相似度计算并排序。由于这400篇文书可能重复的文书，还需要对文书进行一次去重操作。文书推荐的流程图如图3.16所示：



图3.16 文书推荐流程图

由于裁判文书检索与推荐系统的文书集是一个动态演进的过程，如果有新增的文书，则需要对文书推荐表进行更新，保证最新的推荐文书。当有多篇新增的文书时，需要遍历每一篇文书，对文书进行文本预处理，并用LDA主题模型计算出文书的主题分布。将新增的文书主题分布与其他所有的文书主题分布进行相似度计算，如果相似度超过原有文书最相似20篇文书的最低值，将采取淘汰策略，加入新增文书并替换原有最相似20篇的最低值文书。同时，采用淘汰策略不断更新新增文书的最相似20篇文书。依此类推，将所有的新增文书更新到推荐表中，整个文书推荐更新流程图如图3.17所示：



图3.17 文书推荐更新流程图

## 3.5 数据库设计

裁判文书检索与推荐系统主要是对文书信息的检索，为了提高检索效率，仅仅保存文书的文本信息是不够的，还需要对文书的基本特征信息进行提取并且保存到数据库表中。为了避免数据库的冗余，存储数据的表与表之间需要进行适当的信息分离，即将文书的基本信息保存到文书信息表中，将文书的基本属性保存到文书属性表中。当对文书进行条件检索时调用文书属性表，当查看具体一篇文书时，调用文书信息表。其中，文书信息表WsInfo的结构如表3.10所示：

表3.10 文书信息表WsInfo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 描述 | 数据类型 | 长度 | 键类型 |
| WSID | 文书唯一ID | varchar | 32 | PK |
| WS | 文首 | text |  |  |
| DSR | 当事人 | text |  |  |
| SSJL | 诉讼记录 | text |  |  |
| AJJBQK | 案件基本情况 | text |  |  |
| CPFXGC | 裁判分析过程 | text |  |  |
| PJJG | 判决结果 | text |  |  |
| WW | 文尾 | text |  |  |

由文书解析读取xml文书并解析出文书的基本属性信息，将属性信息保存到文书属性表中。文书属性表WsAttribute的结构如表3.11所示：

表3.11 文书属性表WsAttribute

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 描述 | 数据类型 | 长度 | 键类型 |
| WSID | 文书唯一ID | varchar | 32 | PK |
| ah | 文书案号 | varchar | 64 |  |
| wsmc | 文书名称 | varchar | 64 |  |
| wszl | 文书种类 | varchar | 32 |  |
| xmlName | 文书xml文件名 | varchar | 32 |  |
| xmlPath | 文书xml文件路径 | varchar | 32 |  |
| docName | 文书doc文件名 | varchar | 32 |  |
| docPath | 文书doc文件路径 | varchar | 32 |  |
| ajlb | 案件类别 | varchar | 16 |  |
| aycj | 案由层级 | varchar | 16 |  |
| aymc | 案由名称 | varchar | 32 |  |
| aydm | 案由代码 | varchar | 12 |  |
| fycjm | 法院层级码 | varchar | 12 |  |
| bzfymc | 标准法院名称 | varchar | 32 |  |
| fyjb | 法院级别 | varchar | 16 |  |
| cpDate | 裁判时间 | date | 3 |  |

续表 3.11

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 描述 | 数据类型 | 长度 | 键类型 |
| lawer | 律师 | varchar | 32 |  |
| lawFirm | 律所 | varchar | 32 |  |
| isYgct | 原告是否出庭 | tinyint | 1 |  |
| isBgcy | 被告是否出庭 | tinyint | 1 |  |
| cpYear | 裁判年份 | int | 10 |  |
| spcx | 审判程序 | varchar | 16 |  |
| qsf\_gsf | 起诉方，公诉方 | varchar | 32 |  |
| bg | 被告 | varchar | 32 |  |
| spry | 审判人员 | varchar | 64 |  |
| flyj | 法律依据 | varchar | 256 |  |

在用户收藏功能中，用户点击收藏文书时，系统将用户ID和文书ID进行绑定，并将其保存到数据库中，即用户文书表。用户文书表UserWsb的结构如表3.12所示：

表3.12 用户文书表UserWsb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 描述 | 数据类型 | 长度 | 键类型 |
| id | 主键id | bigint | 默认 | PK |
| UID | 用户ID | varchar | 32 |  |
| WSID | 文书ID | varchar | 32 |  |
| createTime | 创建时间 | timestamp | 默认 |  |
| updateTime | 更新时间 | timestamp | 默认 |  |

在文书推荐模块，通过LDA主题模型对每一篇文书计算得到其主题分布，将文书ID和该文书的主题分布保存到数据库中，也就是文书主题分布表。文书主题分布表WsZtfb的结构如表3.13所示：

表3.13 文书主题分布表WsZtfb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 描述 | 数据类型 | 长度 | 键类型 |
| id | 主键id | bigint | 默认 | PK |
| WSID | 文书ID | varchar | 32 |  |
| ztfb | 文书的主题分布 | varchar | 128 |  |
| createTime | 创建时间 | timestamp | 默认 |  |
| updateTime | 更新时间 | timestamp | 默认 |  |

出于推荐效率的考虑，会在后台将文书的相似度较高的推荐文书先计算出来，然后保存到数据库中，当用户查看具体文书需要默认推荐文书时，只要到数据库表中根据文书ID查找到推荐文书列表即可。文书推荐表WsRecommend的结构如表3.14所示：

表3.14 文书推荐表WsRecommend

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 描述 | 数据类型 | 长度 | 键类型 |
| id | 主键id | bigint | 默认 | PK |
| WSID | 文书ID | varchar | 32 |  |
| TJWS | 推荐文书 | varchar | 256 |  |
| createTime | 创建时间 | timestamp | 默认 |  |
| updateTime | 更新时间 | timestamp | 默认 |  |

## 3.6 本章小结

本章主要介绍了裁判文书检索与推荐系统的分析和设计过程，先是描述了系统的用例图，对用户在使用系统中的相关用例进行详细的阐述，包括有文书解析、文书类型转换、文书索引、文书更新、高级检索、文书下载以及文书推荐等功能需求。然后根据需求分析的结果，给出裁判文书检索与推荐系统的整体架构设计，对系统按照模块进行划分，详细介绍了裁判文书检索与推荐系统中各个功能模块的职责以及对应的功能点。最后，介绍了系统中相关的数据库设计，并给出多张表的结构设计。

# 第四章 裁判文书检索与推荐系统的实现

根据上一章对裁判文书检索与推荐系统的需求分析和设计，本章主要通过对文书处理模块、文书管理模块、文书检索模块以及文书推荐模块进行展开，重点描述本项目各个模块的具体实现细节。

## 4.1 裁判文书检索与推荐系统的主要功能概述

裁判文书检索与推荐系统主要目标是为了实现文书的更高效检索，从而帮助用户能够方便快捷检索到想要的相关文书。该系统的主要功能模块是检索与推荐，因此系统中其他功能模块的实现都是为这两个主要功能服务的，帮助系统功能可以完整的实现。本章会对文书相关的功能模块实现进行详细的描述。

文书处理模块包含文书解析和文书类型转换。文书解析是对xml文书的读取并提取文书的基本属性信息，并用对象进行保存，最后再持久化保存到数据库中。文书类型转换是将xml文书转换成doc文书，从文书信息表中读取数据并通过freemarker模板构建doc文书。

文书管理模块包括文书索引以及文书更新。文书索引是将解析好的文书保存到Elasticsearch索引库当中，便于后续高效地检索。文书更新主要对新增文书进行操作，通过文书解析、文书类型转换以及文书索引等功能操作将新的文书信息保存到索引库中。

文书检索模块包括高级检索、文书排序、文书下载以及文书收藏。系统能够通过高级检索即精确条件和模糊条件进行文书的检索，搜索引擎接收到检索条件并开始检索，将检索结果返回给用户。同时，用户可以按照法院层级、裁判日期以及审判程序进行排序。对于感兴趣的文书，用户可以选择不同类型的下载以及收藏，便于下一次直接查看。

文书推荐模块主要是负责具体文书的相似文书推荐。首先需要从数据库中提取训练语料，然后对文本内容进行数据预处理。然后构建一个LDA主题模型，设定主题数。将训练语料放到模型中进行计算训练得到一个适用该应用场景的算法模型。接着通过算法模型对每一篇文书进行主题分布的计算，并将其以键值对的方式保存到数据库中。最后通过计算文书之间主题分布的相似度来推荐相似的文书。当查看具体一篇文书时，系统给出相似的其他文书。

## 4.2 文书处理模块的实现

文书处理模块主要包含文书解析以及文书类型转换两个功能点。该模块主要是对xml文书进行解析并将xml文书转换成doc文书，主要包含的类如表4.1所示：

表4.1 文书处理模块主要类

|  |  |
| --- | --- |
| 分类 | 主要包含的类 |
| 控制类 | AdminController.java |
| 服务类 | WsService.java |
| 数据持久化类 | WsInfo.java |
| 持久化接口 | WsInfoDao.java |
| 工具类 | Dom4jParseXml.java,ConvertWordUtil.java |

文书解析功能中先是通过AdminController控制类来进行总处理，包括调用WsService类来遍历所有xml文书的路径位置，通过readFile方法读取本地服务器上的文书并获取文书的文件名。对于得到的每一篇文书路径，调用Dom4jParseXml类中的Dom4jGetObject方法读取并解析具体一篇xml文书。通过Dom4j解析方式遍历所有节点信息并将提取出的文书基本属性信息封装到WsInfo对象中，将该对象返回给AdminController类，然后将该对象通过WsInfoDao类持久化到数据库中。

在文书解析过程中，需要创建一个SAXReader对象，这是读取xml文件的对象用来指向xml文件的输入流。由于xml文件是磁盘上的一个物理文件，要将其变成一个Java对象需要将文件放到内存中，然后在内存中进行处理。然后创建一个输入流指向xml文件，通过SAXReader对象利用流来进行读取，即将xml文件数据放到内存中进行解析，此时返回一个文档类型的对象。通过该对象，获取xml的根元素，再通过根元素读取子元素，再读取元素点里面的值。由于裁判文书固定的书写格式，每一部分的信息段都有固定的信息数据。比如当遍历到文首节点时，它的子元素一定有案号、文书种类、案件类别、审判程序、经办法院（法院层级码、标准法院名称、法院级别）等文书基本属性信息。在解析过程中，由于可能有几个律师，但都是同一个律师所，所有需要对其进行去重的操作，避免数据的重复。与此同时，由于文书属性表都是字符串的存储格式，需要将一些List类型的数据转化成String类型的数据，各个元素之间用分号隔开。文书解析的相关代码如图4.1所示：

|  |
| --- |
| public WsInfo Dom4jGetObject(String filePath, String fileName){  WsInfo wsInfo = new WsInfo();  //创建SAXReader的对象reader  SAXReader reader = new SAXReader();  try{  String saveFile = filePath + '/' + fileName;  File file = new File(saveFile);  Document document = reader.read(file);  wsInfo.setXmlName(fileName);  wsInfo.setXmlPath(filePath);  Element rootEle = document.getRootElement();  Element QWEle = rootEle.element("QW");  wsInfo.setQw(QWEle.attributeValue("value"));  //逐个字节遍历，提取其中的属性值并set到wsInfo对象中  System.out.print(wsInfo);  }catch(DocumentException e){  e.printStackTrace();  }  return wsInfo;  } |

图4.1 xml文书解析代码

将xml文书中所有相关的关键属性信息全部都提取出来，并用WsInfo对象进行存储，对象的属性名和xml文书节点名称一一对应，xml文书中没有的属性默认设置为null。打印出文书对象的数据如图4.2所示：

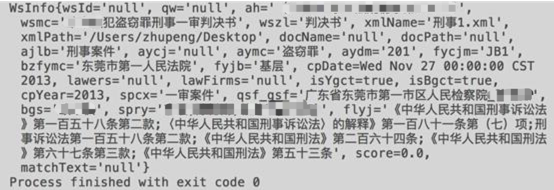


图4.2 文书对象数据

文书类型转换是将xml文书转换成doc文书。由于之前的文书解析已经把数据解析保存到数据表中，因此只需要从数据表中读取数据然后按照文书内容的需要写入制作成word文书就可以了。该功能采用的是freemarker基于word模板导出的方式来制作word文档。转换成的doc文书与xml文书一一对应，都保存在本地服务器上。



图4.3 文书类型转换流程图

如图4.3所示，由ConvertWordUtil类来实现将xml文书转换成doc文书。首先ConvertWordUtil类调用WsInfoDao类将文书信息从数据表中查找出来，返回结果是WsInfo对象。另外新建一个word文档，手动编写好文档的格式，用${variableName}来表示需要文书对象填入的文本，编写结束后另存为Word 2003 XML文档（\*.xml） 格式。修改该文件的后缀名为ftl，使其成为ftl文件，将修改好的ftl文件保存到项目文件的某个路径下。ConvertWordUtil类中的ConvertWord方法负责将WsInfo对象的数据保存转存到ftl文件中。为了防止乱码，以utf-8的编码读取ftl文件。用HashMap的数据类型用来存储需要填入ftl文件的属性名以及属性值。然后将HashMap数据对象写入到ftl文件中，形成xml文件，最后再将xml文件转成系统所需要的word文档。文书类型转换的关键代码如图4.4所示：

|  |
| --- |
| protected void ConvertWord(WsInfo wsInfo, HttpServletResponse response) throws ServletException, IOException {  Map<String,Object> dataMap=new HashMap<String,Object>();  //将wsInfo的相关属性保存到dataMap中  configuration.setDefaultEncoding("UTF-8");  //文件所在的位置  configuration.setClassForTemplateLoading(this.getClass(),"ftl/");  Template t=null;  try {  t = configuration.getTemplate("ws.ftl"); //文件名  } catch (IOException e) {  e.printStackTrace();  }  response.setContentType("application/octet-stream"); response.setCharacterEncoding("UTF-8");  response.setHeader("Content-Disposition","attachment;filename="+  wsInfo.getXmlName() + ".doc");  Writer out = response.getWriter();  try {  t.process(dataMap, out);  } catch (TemplateException e) {  e.printStackTrace();  } catch (IOException e){  e.printStackTrace();  }  } |

图4.4 文书类型转换代码

ConvertWordUtil类是将所有的文书对象全部都查询出来，然后依次遍历文书对象，每转换一次文书都需要调用ConvertWord方法，这样循环转换直到所有的文书全都转换成doc文书。为了保证xml文书和doc文书的一致性，再生成word文档时，文件使用和xml文书一样的名称，只是后缀名不一样。由于在文书信息表中有docName文书名称和docPath文书路径两个属性，在文书解析模块中没有相关的属性值，直接为空，当生成doc文书后，需要将相关的属性值更新到数据库中，维护数据表信息的完整性。由于该功能是为其他功能服务提供的基础支持，不对用户开放，系统需要设置一个默认启动器。当系统启动后检测是否数据存在docName和docPath两个属性值，如果有说明已经转换成word文书。如果这两个属性值为空，说明还没有进行文书转换，调用ConvertWordUtil类将xml文书转换成doc文书。

## 4.3 文书管理模块的实现

文书管理模块主要包含文书索引以及文书更新两个功能。该模块主要是对文书构建索引并对文书进行更新操作，其中包括文书解析、文书类型转换以及文书索引。该模块主要包含的类如表4.2所示：

表4.2 文书管理模块主要类

|  |  |
| --- | --- |
| 分类 | 主要包含的类 |
| 控制类 | AdminController.java |
| 服务类 | WsService.java |
| 数据持久化类 | WsInfo.java |
| 持久化接口 | WsInfoDao.java |
| 工具类 | Dom4jParseXml.java,ConvertWordUtil.java  StringUtil.java,DateUtil.java,TimerTask.java |

文书索引通过读取数据库中的文书信息得到WsInfo对象，再将WsInfo对象转成JSON字符串，将其添加到搜索引擎中。



图4.5 文书索引结构

如图4.5所示，基于Elasticsearch特殊的索引机制，它将数据库中一行文书当作一篇文档，每一个字段当作一个域field。Elasticsearch可以对域中的文本进行分词，使每一个词都能指向含有该词的文档，也可以不对域中的文本进行分词，直接将其指向对应的文档。这样就形成词到文档的倒排索引，在全文检索的时候就不需要遍历所有文书信息来对每个词进行比对，较高地提升了检索的效率。

文书更新功能需要在项目中添加一个定时任务，负责新增文书的定时更新。由于该项目使用的是Spring MVC框架，需要在Spring MVC.xml文件中添加配置，设置定时任务。同时，通过新建TimerTask类来调用需要更新的模块。比如新增文书的解析，文书类型的转换以及Elasticsearch中倒排索引的更新。

## 4.4 文书检索模块的实现

文书检索模块主要包括高级检索、文书排序、文书下载以及文书收藏。用户可以输入检索条件查询到所有相关的文书，在浏览检索结果时，可以按照一定条件进行排序。对于感兴趣的文书，用户可以选择下载或者点击收藏，方便用户下一次的查阅。该模块主要包含的类如表4.3所示：

表4.3 文书检索模块主要类

|  |  |
| --- | --- |
| 分类 | 主要包含的类 |
| 控制类 | SearchController.java,DownloadController.java |
| 服务类 | EsService.java,EsServiceImpl.java |
| 数据持久化类 | WsInfo.java |
| 持久化接口 | SearchDao.java |
| 工具类 | StringUtil.java,DateUtil.java,SortClass.java, SortType.java |

用户进行高级检索时，可以根据系统提示输入检索条件。高级检索分为按条件检索和按全文检索。由于之前文书解析过程中已经提取出文书的基本信息，包括有案由、法院层级、案件类型、年份、审判程序、文书类型等筛选信息，用户可以在下拉框中选择。还有一些信息比如案件名称、案号、法院名称、法律依据等输入信息，用户可以自行进行输入。当用户点击检索按钮，系统会根据用户的查询条件查找符合用户条件的文书，并将结果返回给用户查看。高级检索的关键代码如图4.6所示：

|  |
| --- |
| @RequestMapping(value = "/complexSearch")  public String complexSearch() {  HashMap<String, String> preciseConditions = new HashMap<>();  HashMap<String, String> ambiguousConditions = new HashMap<>();  //将检索条件相应的存入preciseConditions和ambiguousConditions中  List<WSInfo> list = esService.getWSInfoList();  int count = 0;  count = (int) esService.getWSInfoListNum();  int maxPageNum = 0;  //分页处理  model.addAttribute("maxPageNum", maxPageNum);  return "search";  }  Public List<WSInfo> getWSInfoList() {  List<WSInfo> list = new ArrayList<WSInfo>();  BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();  //将检索条件存入BoolQueryBuilder对象中  HighlightBuilder highlightBuilder = new HighlightBuilder();  //设定检索域  SearchResponse scrollResp;  //根据Java官方API进行条件查询  return list;  } |

图4.6 文书检索部分代码

用户根据系统提供的检索项来进行查询，按照给定的要求输入检索条件。有些条件需要文本输入，有些条件需要进行下拉框选择，文书高级检索实现界面如图4.7所示：



图4.7 文书高级检索界面

系统根据用户的检索条件得到文书集，并将文书结果在系统界面上显示。系统的搜索结果界面分为三个功能区域，第一区域是文书结果的展示，以列表的形式将所有的文书展示出来。展示信息包括文书的名称，检索条件匹配的相关语段，还包括单篇文书的直接下载和推荐文书功能。第二区域是用户的继续检索区域，用户可以重新选择检索条件进行文书检索，这区域是对首页系统界面检索的延续，方便用户重新进行文书检索。第三区域是对检索文书结果的二次筛选，包括多种类目，比如按关键字筛选、按案由筛选、按法院层级筛选、按地域及法院筛选、按裁判年份筛选、按审判程序筛选以及按文书类型筛选。通过上述条件的输入，能够方便用户得到相应的一组文书数据。用户可以任意组合筛选条件，直到不存在符合所有条件的文书为止。与此同时，用户可以对感兴趣的文书进行批量收藏，当点击批量收藏后，系统会绑定用户ID以及要收藏的文书ID并提示用户收藏成功。系统检索结果的界面如图4.8所示：



图4.8 文书检索结果列表界面

文书下载功能主要是由DownloadController类来实现，其中包括单篇文书的下载和多篇文书的下载。在浏览文书列表界面，用户可以选择下载多篇文书，或者单篇文书，也可以选择下载类型xml文书或者doc文书。当在下载全部文书时DownloadController类中的downloadAll方法会接收前台传入的检索条件，同样需要对条件进行处理。用户通过下拉框选择的检索条件存放到模糊条件preciseConditions中，用户输入的检索条件全都存放到ambiguousConditions中。然后调用ESServiceImpl类的generateZip方法，该类是接口ESService的实现类。generateZip方法主要是从搜索引擎中查询出文书结果并对文书结果进行压缩。主要代码如图4.9所示：

|  |
| --- |
| Public File generateZip(HashMap<String,String> preciseConditions, HashMap<String, String> ambiguousConditions, String beginDate, String endDate, String fileName, String type) {  BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();  //将检索条件存放到BoolQueryBuilder对象中  //构造zip压缩包  File zipFile = new File(fileName);  FileOutputStream fileOutputStream = null;  ZipOutputStream zipOutputStream =  new ZipOutputStream(fileOutputStream);  //进行压缩存储  zipOutputStream.setMethod(ZipOutputStream.DEFLATED);  //压缩级别设置为1(可取值0-9，级别越高，压缩比越高，用时越长)  zipOutputStream.setLevel(1);  //获取查询构造器  int count=0;  SearchResponse scrollResp = searchRequestBuilder.get();  do {  for (SearchHit hit : scrollResp.getHits().getHits()) {  //Handle the hit...  String path = (String) hit.getSource().get("XMLPATH");  count++;  File tmp = new File(getSingleRealFilePath(path, type));  }  scrollResp=transportClient.prepareSearchScroll(scrollResp.getScrollId()).  setScroll(new TimeValue(SCROLL\_TIME)).execute().actionGet();  } while (scrollResp.getHits().getHits().length != 0);  return zipFile;  } |

图4.9 文书下载主要代码

文书下载包括xml文书下载和doc文书下载。用户在检索之后，系统会返回文书列表。用户可以选择一篇文书下载，也可以选择多篇文书下载。当下载多篇文书时，即批量下载，系统会将所有要下载的文书打包，以下载时间重新命名。如果只有一篇文书需要下载，无论是xml格式还是doc格式，文书名称和原来保持一致。在推荐文书列表中，用户依然可以选择单篇下载或者多篇下载，也可以选择不同文书类型进行下载。在查看具体一篇文书时，用户可以直接选择下载单篇文书。当系统下载好所有的文书时，会提示用户下载完毕，文书下载界面如图4.10所示：

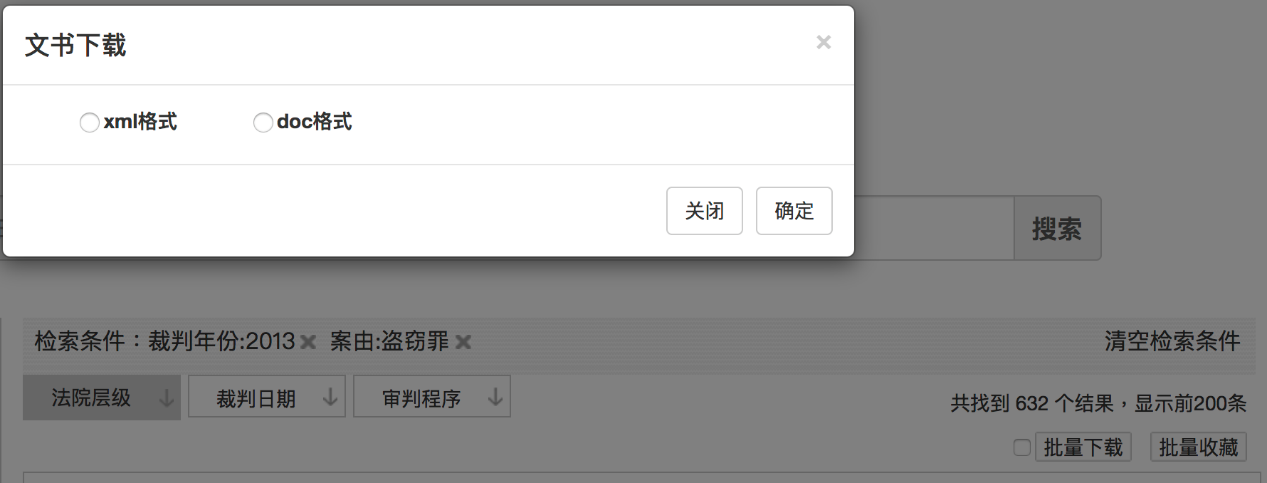


图4.10 文书下载界面

文书检索结果界面中用户可以在每一页浏览十篇文书列表信息，当用户点击具体某一篇文书时，系统会跳转到查看详细文书页面。包括文书名称以及一些关键的基本属性信息，比如：法院名称、案由、案件类型、文书类型、审判程序、裁判日期等信息。然后下面就是文书的全部正文，主要包括当事人、诉讼记录案件基本情况、裁判分析过程、判决结果以及文尾等文本信息。然后右边栏目是默认的推荐文书，系统界面如图4.11所示：

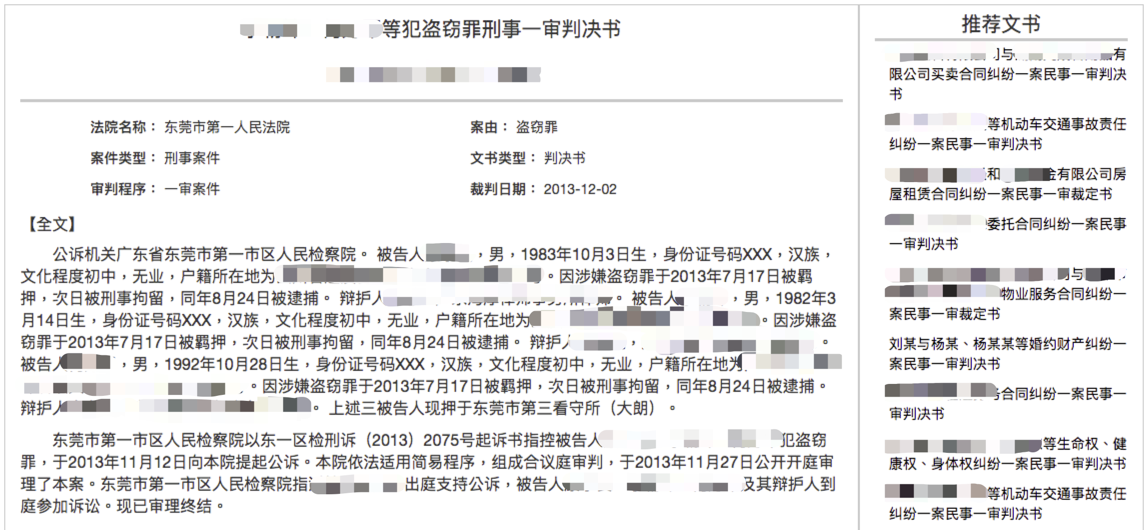


图4.11 文书查看界面

## 4.5 文书推荐模块的实现

文书推荐模块是在用户查询具体一篇文书时，给用户推荐相似度较高的相关文书。推荐的标准是基于裁判文书主题分布相似性的原则，即如果两篇文书的主题分布相似性很高，会认为这两篇文书具有较高的相似性。文书推荐模块主要分为三个部分：LDA主题模型的训练、相似文书推荐以及推荐效果的评估。

### 4.5.1 LDA主题模型的训练

LDA主题模型的训练主要包括文书数据的预处理以及模型的训练。其中文书数据的预处理包括抽取文书的文本信息，对文本进行分词，通过专有的法院停用词表去除无关词语，最后文书信息通过词语向量来表示。由于LDA主题模型只有一个主题数k需要设定，通过训练集对主题模型进行训练，并用测试集对训练得到的模型进行评估，最后得到主题模型的最优主题数。模型训练的流程图如图4.12所示：



图4.12 LDA主题模型训练流程图

数据的预处理是为了获得与训练模型相关的内容，去除噪声数据，以此来提高LDA主题模型的训练效果。预处理主要包括裁判文书的语料提取、文中内容中文分词、构造文书专有提用词库以及去除停用词四个步骤。裁判文书是半结构化的文本结构，一篇结构完整的裁判文书主要是由文首、当事人、诉讼记录、案件基本情况、裁判分析过程、判决结果和文尾组成。对于一篇文书而言，其主要特征是由文首、案件基本情况和判决结果的内容构成的，如果文书的这几个部分内容相似的话，基本可以判定两篇文书几近相似。所以要先将文书的这几个字段信息提取出来作为训练模型的语料，这样能够减少噪音数据，提高训练模型的准确性，降低复杂度。然后使用结巴分词对文本信息进行文字分割，将一篇文书分割成一个一个的词语，然后对其进行使用corpus来读取数据信息，构建所有文书的语料库。与此同时，需要对每一篇文书进行分词处理，去除停用词，然后将处理好的词语用语料库的词来表示。也就是语料库中的词都有固定的序号位置，一篇文书的所有词都用序号来进行表示，方便后面主题模型的训练。在训练模型的过程中，将用矩阵进行表示的文书投入到模型中进行训练，主题模型需要先行构建，设定主题数。通过对所有文书的计算迭代能够得到拟合当前应用场景的主题模型。主题模型的训练相关代码如图4.13所示：

|  |
| --- |
| def load\_stopwords(self):  stopwords = [line.strip() for line in  open('stopwords.txt', encoding='UTF-8').readlines()]  return stopwords  def dataClean(self, text):  # 简单分词  text\_depart = jieba.cut(text, cut\_all=False)  # 创建一个停用词列表  stopwords = self.load\_stopwords()  # 输出结果为outstr  # 去停用词  return outstr  def train():  path = Settings.DATSET\_FILE  texts = readAll(path)  texts\_clean = [[word for word in seg\_depart(text).split() if not len(word) <= 1]  for text in texts]  texts\_complete = remove\_once(texts\_clean)  # 创建语料的词语词典，每个单独的词语都会被赋予一个索引  dictionary = corpora.Dictionary(texts\_complete)  dictionary.save('dictionary.dict')  corpus = [dictionary.doc2bow(text) for text in texts\_complete]  corpora.MmCorpus.serialize('corpus.mm',corpus) |

图4.13 主题模型训练相关代码

### 4.5.2 相似性文书推荐

在得到主题模型之后，需要再次遍历所有的文书，同样对数据进行预处理，进行分词，去停用词等相关的数据操作。在训练LDA模型的时候会根据所有文书分词的集合去重后得到一个字典，所有的分词都在字典当中并且拥有一个唯一的位置值。然后通过字典将文书构建成一个稀疏矩阵。这个稀疏矩阵的元素都是键值对构成，其中键表示该词在字典中的位置，值表示该词在文书中出现的次数，这样一篇文书的完整信息就能够通过稀疏矩阵来进行表示。然后将这个矩阵投入到训练好的主题模型中进行计算，得到文书的主题分布。假定LDA模型设定的是20个主题数，则文书通过模型计算得到的结果就会是20维的向量，每一个维度代表一个主题所占的比重。这样文书的主题分布能够通过向量来进行表示，文书主题分布的数据如图4.14所示：

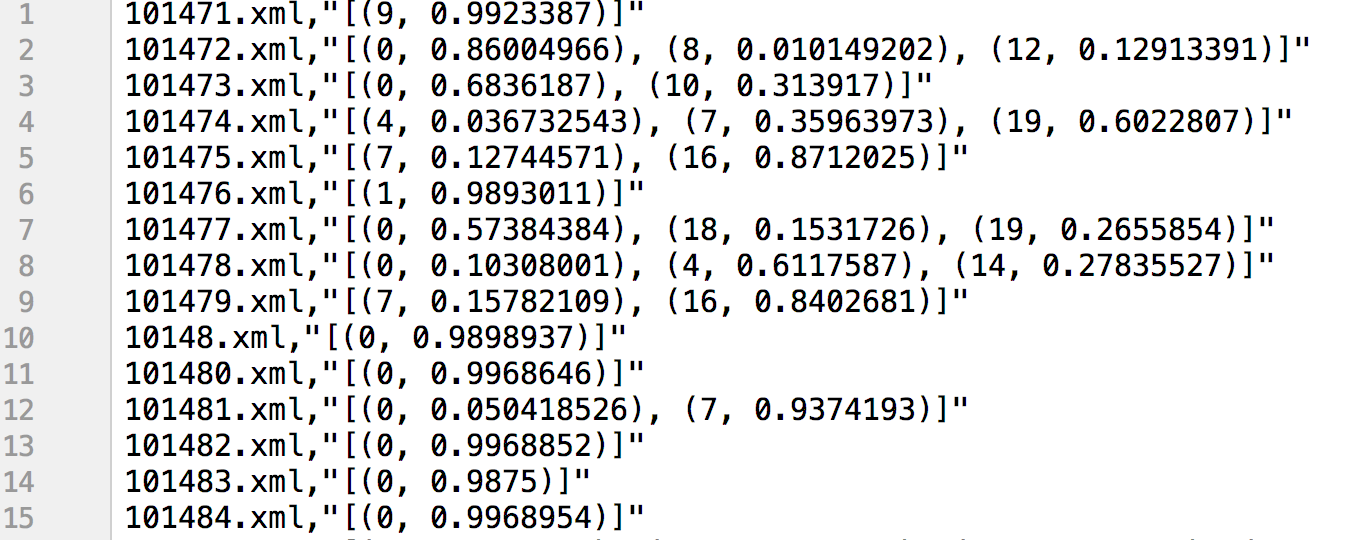


图4.14 文书主题分布数据图

文书的主题分布是文书文本向量空间的映射，可以通过训练得到的主题模型得到文书的主题表示，因此文书之间的相似度可以通过计算文书主题概率分布的距离来实现。由于主题是词向量的混合分布，可以通过JS分歧算法来度量文书之间的相似度。JS分歧度越小，表示两篇文书的主题概率分布越接近，从而能够推测出文书越相似。

在项目工程实践中，当用户查看具体一篇文书需要查看类似的文书时，需要将其他所有的文书与目标文书进行分歧度计算，然后取分歧度数值小的前20篇文书推荐给用户。这样的操作方法固然总是能够将最相似的文书推荐给用户，但每次推荐文书都需要遍历所有文书的相似度，系统的运行效率会比较低。因此采用一个折中的方法，先对每篇目标文书进行计算得到它的主题分布分歧度最低的前20篇文书。以键值对的形式保存到数据库中，key代表的是文书的ID，value代表的20篇文书的ID和对应的分歧度。这样，当需要推荐相似的文书给用户时只需要根据这个表找到要推荐文书的ID就直接给出相似的20篇文书。如果用户想要知道更多相似文书的话，假定认为目标文书的相似文书，它们的相似文书也必然和目标文书相近，类似于广度遍历的思想。将相似文书的20篇文书与目标文书进行分歧度计算。文书推荐的计算流程如图4.15所示：



图4.15 文书推荐的计算流程图

### 4.5.3 推荐效果评估

裁判文书检索与推荐系统是基于LDA主题模型的文本相似度推荐，先是训练好适用于应用场景的主题模型，再根据主题模型得到主题分布并计算主题分布的相似度。因而在评估过程中分为两个阶段，一是主题模型好坏的评估，二是基于主题分布相似度推荐效果的评估。

LDA模型的好坏一定程度影响了推荐效果，因此对于主题模型的评估能够很大程度的影响推荐效果。由于LDA模型是非监督的主题模型，在LDA模型的训练过程中，一个重要的参数设定是主题数量K。如果主题数量过大，主题粒度就会过细，可能造成模型过拟合，泛化能力差。反之如果主题数量过小，主题粒度就会过粗，文本的语义信息将无法得到很好的表达。本节通过Perplexity（困惑度）来反映主题模型的泛化能力，并以此指标评价模型的优劣。困惑度是指文本在划分主题时确定性的评判标准，反映的是模型对于新样本数据的适用性。其计算方式如公式（4.1）所示：

公式（4.1）

其中M表示文书集的数量，表示第m篇文书中的词语个数，表示为第m篇文书中词的概率值（所有词的概率乘积）。主题数量越多，困惑度越小，但是模型越容易过拟合。在训练评估模型中，将75%的文书作为训练集以训练模型，25%的文书作为测试集用来计算困惑度，以反映模型的泛化能力，最后得到最优的主题数。

图4.16 困惑度趋势图

由图4.16所示，随着主题个数k越来越大，困惑度也越来越小，但主题数60开始往后困惑度下降越来越趋于0。考虑到随着主题数的增加，LDA模型会容易产生过拟合，同时系统的计算负荷也越来越高，因此选择主题数为60最为合适，既考虑到主题模型的泛化能力又兼顾了系统的计算能力。

在评估基于LDA主题分布的相似度文本推荐时，引入另外一种文本相似度计算方法：TF-IDF模型。文书中每个词的权重由TF\*IDF表示，其中TF表示词频即一个词在文书中出现的频率，IDF表示逆文档频率即一个词在所有文书中出现频率的倒数。因此，一个词在该文书中出现越多，在其他文书中出现越少，表示这个词很好地反映文书的内容，权重就越大。TF-IDF计算方式如公式（4.2）和公式（4.3）所示：

公式（4.2）

公式（4.3）

每一篇文书都可以用TF-IDF来进行表示，然后用余弦相似度来计算文书之间的相似度并给出推荐的相似文书集。余弦相似度是用向量空间中两个向量夹角的余弦值作为衡量两个个体间差异的大小的度量。余弦值越接近1，就表明夹角越接近0度，即两个向量越相似。

从业务层面看，两篇文书的相似与否主要是看案由以及文书中的相关内容。所以需要设定一个相似指数，相似指数的计算方式如公式（4.4）所示：

公式（4.4）

其中，假定案由以树状形式展现，案由距离是指一个案由到另外一个案由的距离。比如两个案由相同，案由距离为0；一个案由是另外一个案由的上层案由，案由距离为1；同一个上级案由下的两个案由距离是2，以此类推。文书相似度是通过文本的余弦相似计算得出的，文本相同取值为1，完全不同取值为0。当两篇文书完全相同时，相似指数为1。

通过相似指数标识出目标文书的正确推荐文书，将文书推荐的正确与否看出是一个二分类，通常采用召回率（Recall）与准确率（Precision）作为评估标准。以二分类为例，其混淆矩阵如表4.4所示：

表4.4 混淆矩阵

|  |  |  |
| --- | --- | --- |
| 预测  真实 | 正 | 负 |
| 正 | TP | FN |
| 负 | FP | TN |

准确率如公式（4.5）所示：

公式（4.5）

召回率如公式（4.6）所示：

公式（4.6）

F1值为准确率和召回率的平衡标准，如公式（4.7）所示：

公式（4.7）

在推荐文书的过程中，当模型推荐较少文书时，正确推荐的文书数量占总推荐文书比例较高，此时准确率较高，但召回率较低。随着推荐文书的增多，正确推荐的文书占总推荐文书的比例慢慢降低，但召回率不断增高。如果只关注准确率，系统推荐较少的文书就能得到较高的准确率，但推荐文书增多的效果则不能够衡量。如果只关注召回率，系统推荐大量的文书时，召回率会很高，但也会推荐很多不相关的文书。通过对F1的评估，既能够衡量系统推荐文书的准确性又能衡量推荐更多文书时的效果。

由于每篇目标文书对应的正确推荐文书数目都不相同，会导致召回率受到影响，从而影响到F1值。在评估过程中，随着推荐文书数目的变化，对所有文书的准确率、召回率以及F1值求平均值。

表4.5 LDA和TF-IDF在不同推荐文书下F1值的对比

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 20 | 40 | 60 | 80 | 100 |
| LDA | 39.4% | 48.6% | 55.7% | 63.6% | 60.8% |
| TF-IDF | 37.3% | 35.5% | 51.3% | 59.4% | 56.6% |

推荐效果如表4.5所示，在推荐20篇文书时，两个模型的F1值都比较低，这是因为目标文书的正确推荐文书较多，此时准确率虽然很高，但召回率很低，导致两个模型的F1值很低。随着推荐文书数量的不断增加，准确率慢慢降低，但召回率在不断增高，导致两个模型的F1值不断增高。当推荐的文书不断接近正确的推荐文书数量时，此时，推荐的准确率和召回率都趋于平均，两个模型的F1值达到最高值。当推荐文书继续增加时，召回率会越来越高慢慢趋近100%，但准确率又慢慢变低，情况和推荐文书较少时相反，此时F1值会慢慢减小。由于每篇目标文书的正确推荐文书都不相同，所以每篇文书推荐效果最好的推荐数目也不相同，这里是对所有的目标文书取的平均值，当推荐文书数目达到80篇时，推荐效果最好，并在之后缓慢减低。并且在不同推荐数目下，LDA模型的F1值都要比TF-IDF模型下的F1值要高，从而能够得出结论：基于LDA主题模型的文本相似度推荐有着良好的效果。

在这一小节，先是通过困惑度来验证LDA模型并得到最优主题数，然后通过F1值来衡量LDA模型和TF-IDF模型下的推荐效果，从而证明基于LDA主题模型文本相似性的推荐方式有更好的表现效果。

## 4.6 裁判文书检索与推荐系统的测试

### 4.6.1 测试思路

软件测试的目标在于发现软件中的缺陷和不足，验证软件是否满足需求中提出的功能要求。软件测试是为评价与改进被测试的软件的质量、发现缺陷和问题而进行得活动[袁帅，2014]。通过对本系统测试希望达到以下目标：

（1）确保系统能够正常稳定的运行。

（2）确保系统的运行效率能够满足用户的体验感。

### 4.6.2 性能测试

性能测试的目的在于验证系统在高负载的运行环境下是否能够保持正常的工作，一般通过自动化的测试工具来模拟高负载的条件，在极限条件下测试系统的运行指标。

对于系统的文书检索模块和文书推荐模块，分别对应uc1、uc2。这两个功能模块会对用户的系统体验产生直接的影响，因此需要有较高的性能，本文使用JMeter测试工具来评估系统处理数据的能力，使用平均响应时间来衡量系统的处理能力。

文书检索模块和文书推荐模块的性能测试代码类似，因此本文只给出检索模块的相关代码，详细的代码如图4.17所示：

|  |
| --- |
| ..导出的相关包  public class Dom4jParseXmlTest implements JavaSamplerClient {  private SampleResult results;  ..设置默认参数  ..初始化方法，运行前执行  /\*\*  \* 执行测试  \*  \* @param javaSamplerContext  \* @return  \*/  @Override  public SampleResult runTest(JavaSamplerContext javaSamplerContext) {  ..获取在Jmeter中设置的参数值  //开始统计响应时间  results.sampleStart();  try {  //被测对象调用  esService. getWSInfoList(preciseConditions, ambiguousConditions,  beginDate, endDate, isDefault, order, beginIndex, istNum);  results.setSuccessful(true);  } catch (Throwable e) {  results.setSuccessful(false);  logger.debug(e);  } finally {  //结束统计响应时间  results.sampleEnd();  }  return results;  }  ..测试方法运行结束后执行  public static void main(String[] args) {  //设置参数  Arguments params = new Arguments();  ..  JavaSamplerContext arg0 = new JavaSamplerContext(params);  PerformenceTest test = new PerformenceTest();  test.setupTest(arg0);  test.runTest(arg0);  test.teardownTest(arg0);  }  } |

图4.17 文书检索模块测试代码

本文采用的压测方法为梯度压力测试，通过逐渐增加并发数，获得系统的相关性能数据。详细的测试数据如图4.18所示，当系统的并发数为小于400时，检索模块和推荐模块的响应时间均小于850ms，当系统的并发数为800时，检索模块和推荐模块的响应时间均小于950ms。由于系统高并发的应用场景不多，正常情况下并发数保持在800以下，因而能够保证系统在检索和推荐方面低延迟的运行效果。综上所述，系统的性能基本满足了用户的需求，能够给用户提供较好的系统体验感。

图4.18 并发数-平均响应时间折线图

## 4.7 本章小结

本章在第三章设计与分析的基础上主要讲述了裁判文书检索与推荐系统相关模块的实现细节，包括文书处理模块、文书管理模块、文书检索模块以及文书推荐模块。并且对推荐模块的算法模型进行评估，确定了使用基于LDA模型主题分布相似度推荐方法的可行性。最后对系统的部分核心模块进行性能测试，保证了系统的运行效率。

# 第五章 总结与展望

## 5.1 总结

裁判文书检索与推荐系统主要是帮助用户更高效地检索想要的文书，能够进行较为复杂的条件检索。对于从事司法研究的科研人员，该系统能够帮助他们检索到大量有效的文书数据以支持科研工作。在不确定详细检索条件的情况下，基于文书主题分布相似性的推荐能够帮助用户更加方便地检索到一系列相关文书。本人有幸负责裁判文书检索与推荐系统的开发，包括文书处理、文书管理、文书检索以及文书推荐功能模块的实现。

本文主要基于裁判文书检索与推荐系统的设计与实现进行展开，并对文书检索的相关模块进行了重点描述。本文首先介绍了文书检索的行业背景，对传统全文检索技术的发展过程进行了介绍，并描述了裁判文书检索与推荐系统的项目背景和项目概述，阐述了裁判文书检索与推荐系统主要解决的问题和需要提供的相关服务。然后介绍了裁判文书检索与推荐系统所使用的相关开发技术，包括整个项目采用的MVC设计思想，Elasticsearch搜索引擎、LDA主题模型以及Spring框架等多种技术。紧接着描述了裁判文书检索与推荐系统的功能需求和文书检索相关模块的分析与详细设计，并对文书信息存储以及文书推荐所涉及的数据库表的设计进行了介绍。接着通过流程图和时序图直观的描述文书检索相关模块的实现逻辑，并且通过部分关键代码描述了裁判文书检索与推荐系统中文书相关模块的实现方式，并通过运行的系统部分截图展示了功能模块的实现效果。最后，根据第三章和第四章的系统需求描述和功能实现进行相关的系统测试，编写测试用例，执行测试验证系统的执行结果是否和测试用例期望值一致。

裁判文书检索与推荐系统基于Elasticsearch搜索引擎来进行全文检索，不同于以往传统的关系型数据库内容，需要遍历全文一一进行比对。Elasticsearch采用的是倒排索引的方式，通过词语就能够直接索引到文书ID，这样大幅度提高了文书的检索速度。推荐功能是基于LDA主题模型的文本相似度的计算，根据分歧度的大小来进行文书的推荐，而不需要对文书内容进行一一的比对，大幅度提高了文书推荐的效率。

## 5.2 进一步工作展望

目前裁判文书检索与推荐系统已经大致完成，用户能够输入检索条件进行文书的搜索，并依据查看的文书内容推荐相似的文书。但依然存在一些不足：

（1）作为一个文书检索系统，功能较为单一，可以增加一个用户登录功能，针对用户的检索习惯做出个性化的推荐。

（2）训练的LDA主题模型较为的粗糙，数据预处理中可以加入词向量的训练，比如TF-IDF以及word2vec，使得模型更加的拟合应用场景。

# 参考文献

[Bai, 2013] J. Bai, Feasibility analysis of big log data real time search based on Hbase and ElasticSearch, In 2013 9th International Conference on Natural Computation (ICNC’2013), pages 1166-1170, 2013.

[Blei et al., 2003] D. M. Blei, A. Y. Ng, and M. I. Jordan, Latent Dirichlet Allocation, *Journal of Machine Learning Research,* 3(2003): 993-1022.

[Kashiji et al., 2004] S. Kashiji, E. Atlam, and M. Fuketa, A high-speed dynamic full-text search method by using memory management, *International Journal of Computer Mathematics, 2004, 81(12)*: 1477-1492.

[Lei et al., 2015] X. F. Lei, Z. Wang, and Y. Z. He, The Data Management and Real-time Search Based on Elasticsearch, In *Computer Science and Electronic Technology International Society (ICCMCEE’2015)*, pages 825-829, 2015.

[Taylor et al., 2018] R. Taylor, M. H. Ali, and I. Varley, Automating the processing of data in research. A proof of concept using Elasticsearch*,* In *International Journal of Surgery*, pages S41-S41, 2018.

[Wang et al., 2011] C. Wang and D. M. Blei, Collaborative topic modeling for recommending scientific articles, In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 448-456, 2011.

[Wang et al., 2017] Y. Wang, J. Ge, and Y. Zhou, Topic Model Based Text Similarity Measure for Chinese Judgment Document, *Data Science*, 2017:42-54.

[Rosen-Zvi et al., 2004] M. Rosen-Zvi and T. L. Griffiths, The author-topic model for authors and documents. *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, 2012:487-494.

[陈雄华等，2012] 陈雄华，林开雄，*Spring3.X企业应用开发实战*，电子工业出版社，2012。

[陈睿等，1991] 陈睿，陈光祚，谢新洲，湖北省地方志全文检索系统 (上)，*情报理论与实践*，1991(2):34-35。

[曾亚飞，2016] 曾亚飞，*基于Elasticsearch的分布式智能搜索引擎的研究与实现*，硕士论文，重庆大学，2016。

[黄睿，2007] 黄睿，*Spring深度整合指南*，电子工业出版社，2007。

[韩冰，2009] 韩冰，*垂直搜索引擎个性化推荐研究与应用*，硕士论文，大连理工大学，2009。

[何巨璞，2014] 何巨璞，*全文检索中索引算法的优化研究*，硕士论文，重庆大学，2014。

[胡琼，2001] 胡琼，基于WWW的全文检索系统性能探讨，*情报科学*，2001，19(6):639-642。

[宏朴, 2016] 宏朴，*基于Lucene的搜索引擎的研究与实现*，硕士论文，大连理工大学数学科学学院，2016。

[马超等，2016] 马超，于晓虹，何海波，大数据分析：中国司法裁判文书上网公开报告，*中国法律评论*，2016(4):195-246。

[马鹏杰，2018] 马鹏杰，*基于Elasticsearch的地名和POI数据检索系统的设计与实现*，硕士论文，西安电子科技大学，2018。

[蒲奇等，2008] 蒲奇，许林英，Spring框架中Ioc的实现，*微处理机*，2008，29(1):147-149。

[钦蒋承等，2018] 钦蒋承，沈宏良，基于Elasticsearch的校内全文搜索平台的研究与实现，*现代计算机*，2018，634(34):98-102。

[向李兴，2015] 向李兴，*基于自然语言处理的裁判文书推荐系统设计与实现*，硕士论文，南京大学，2015。

[肖运文，2016] 肖运文，*基于Elasticsearch的教育资源推荐系统设计与实现*，硕士论文，北京工业大学，2016。

[徐戈等，2011] 徐戈，王厚峰，自然语言处理中主题模型的发展，*计算机学报*，2011, 34(8):1423-1436。

[薛峰等, 2012] 薛峰，梁锋，徐书勋，王彪任，基于Spring MVC框架的Web研究与应用， *合肥工业大学学报(自然科学版)*，2012，35(3):337-340。

[杨凡，2018] 杨凡，*基于LDA主题模型的在线评论聚类分析与推荐*，硕士论文，大连理工大学，2018。

[袁帅，2014] 袁帅，*电信融合计费账务处理系统的设计与实现*，硕士论文，东南大学，2014。

[袁绪峰，2006] 袁绪峰，基于Spring框架的AOP编程，*计算机与现代化*，2006(1):118-120。

[张高伟，2014] 张高伟，*基于Elasticsearch的分布式视频垂直搜索引擎的设计与实现*，硕士论文，哈尔滨工业大学，2014。

[张卫丰等，2000] 张卫丰，徐宝文，Web搜索引擎框架研究，*计算机研究与发展*，2000，37[3]:376-378。

[周嘉程，2018] 周嘉程，*基于Spring Boot的在线文献管理系统的设计与实现*，硕士论文，南京大学，2018。

[周燕玲，2016] 周燕玲，Spring MVC框架开发Web应用程序的探索与研究，*科技广场*，2016(6):25-28。

# 致 谢

时间飞逝，我在南京大学软件学院两年的研究生学习生活即将结束。在这个过程中，项目组的老师和周围的同学都在学习上给予我巨大的帮助，在此表示衷心的感谢。

首先要感谢指导我毕业设计以及论文撰写的葛季栋老师。葛老师治学严谨，认真负责。无论是前期的课题选择还是后面的学习工作指导，都给我莫大的帮助。在做毕业设计和撰写论文的过程中多次询问我的进展，并指出不足以及给出指导性的意见。同时还要感谢周筱羽老师，在我刚踏进研究生时给予的帮助以及对我的鼓励。

感谢我周围的同学们，在平时的学习生活中给予我的帮助。在做毕业设计的时候遇到问题，通过与同学之间的沟通交流总是能帮助我找到解决的方案。在这近两年的学习过程中帮助我不断地提升专业技能，丰富自己的知识储备。

感谢我的家人，是他们在背后一直无私地支持我不断前进。

最后，感谢各位老师在百忙之中花时间评审我的论文。

# 

# 版权及论文原创性说明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任。

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名：

日期： 年 月 日