



商务智能期末项目报告

1753837 陈柄畅

1650985 郁佳麟

1753771 蔡奕阳

1754266 贺鹏宇

指导老师：朱宏明

目录

1	项目概述	4
1.1	基础查询	4
1.2	智能分析	4
1.2.1	业务场景	4
1.2.2	主要功能	4
2	ETL	6
2.1	数据集	6
2.2	数据清洗	6
2.3	数据导入	6
2.4	知识图谱构建	7
3	Storage Server	9
3.1	数据库选择	9
3.2	数据库详情	9
3.2.1	Neo4j	9
3.3	查询功能实现	9
3.3.1	单节点及其关系查询	9
3.3.2	两节点间多条关系查询	9
3.3.3	两节点间最短路径查询	10
3.4	查询性能优化	10
4	BI Server	11
5	UI	16
5.1	智能分析	16
5.1.1	歌手搜索界面	16
5.1.2	歌手搜索结果	16
5.1.3	歌手详情及推荐	17
5.1.4	推荐歌手对比	18
5.2	基本查询	19
5.2.1	基本查询界面	19
5.2.2	查询结果	19
5.2.3	拓展查询	20

6	加分项总结	22
6.1	ETL	22
6.1.1	如何支持更大规模的数据?	22
6.1.2	如何更好地支持数据更新?	22
6.2	Storage Server	22
6.2.1	如何更好地对图数据库建模?	22
6.2.2	如何提高查询性能?	22
6.2.3	如何增强系统的可扩展性	22
6.3	BI Server	22
6.3.1	如何提高查询的性能?	22
6.3.2	是否可以采用缓存?	23
6.3.3	如何向通用的数据中台靠拢?	23
6.4	可视化	23
6.4.1	如何让用户更好地查看检索的结果? 是否支持沿着图进行进一步扩展?	23

项目概述

本项目可以分为基础查询和智能分析两个部分。

1.1 基础查询

基础查询模块为用户提供两个查询服务

1. 输入一个实体，查询其关联的所有关系和关联实体。可筛选所关联的其他实体的跳数以及数量。
2. 输入两个实体，查询其可能存在的多跳关系。可筛选多跳关系的跳数或是否为最短路径以及返回数量。

同时，基础查询模块为以上两个查询的结果提供可视化展示，不同类别的实体对应不同的图标，且支持沿图进一步拓展。

1.2 智能分析

1.2.1 业务场景

我们基于 DBPedia 的实体-关系数据集以及其他外部数据，设计了基于相似歌手匹配与对比的商务智能业务场景。

主要规划了以下 2 个业务场景：

1. 商演主办方在选择邀请歌手时，将目标歌手与其相似歌手进行对比，选择最为合适的歌手。
2. 经纪公司在签约歌手时，需要考虑该歌手与其他歌手之间的相似以及竞争关系，从而综合各方面因素，做出决策。

1.2.2 主要功能

基于我们设想的业务场景，我们再智能分析模块实现了以下 3 个主要功能：

1. 根据用户输入的关键词，模糊查询歌手。

2. 对于选定歌手，展示与其相关或相似的歌手。
3. 在相关或相似的歌手之间，展示不同类型的数据（文字、图片、视频等），提供多维度的对比。

ETL

2.1 数据集

本项目的知识图谱来自于源于维基百科（Wikipedia）的结构化数据抽取（DBPedia）。在此数据集中，每一行代表一个关系元组。关系元组由主语，谓语，宾语组成。经过分析，由本数据集构建而成的总知识图谱包括 5900560 个实体节点和 18746174 条关系。

2.2 数据清洗

在做数据清理时，我们发现关系元组中的每个实体与关系都有额外的 `<xml>` 标签，在进行数据导入之前需要进行清洗。经过对源数据的分析，我们发现：

对于所有的关系，其 `<XML>` 标签的形式均为：

1. `<http://dbpedia.org/ontology/relationship>`

因此在数据清洗时只需要获取 `url` 最后代表关系的单词即可。

而对于包括主语和宾语的实体，其 `<XML>` 标签的形式分为两种：

1. `<http://dbpedia.org/resource/entity>`
2. `<url>`

前者代表某一实体，后者则具体指向某一网站的连接。对于前者，清洗方法与关系相同，取 `url` 最后代表实体的单词即可。对于后者，只需清洗掉左右两侧的 `<XML>` 标签即可

2.3 数据导入

在数据导入阶段，我们采用的方法为首先将所有的三元组导出为 `csv` 文件，然后将 `csv` 导入 `neo4j` 数据库中。在将 `csv` 文件导入数据库中时，我们共有两种方案：

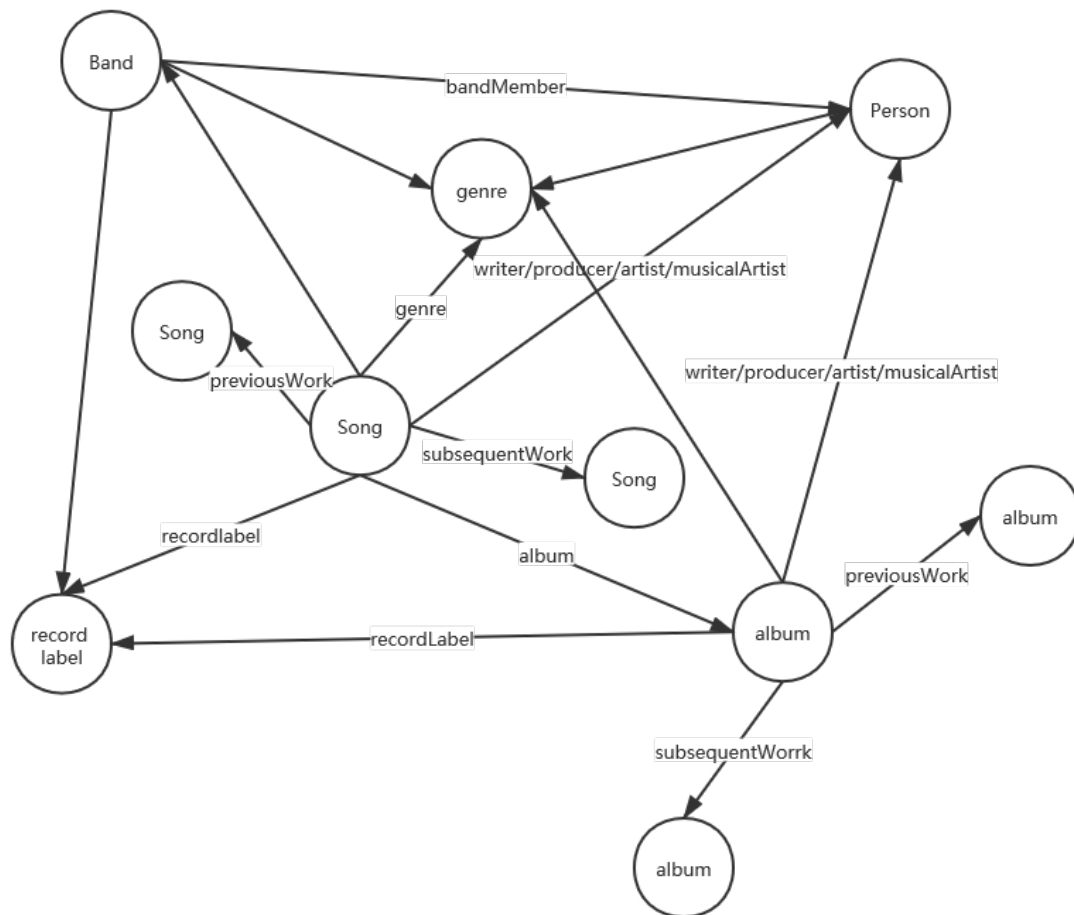
1. `neo4j admin-import`: `neo4j` 官方出品的 `etl` 工具，导入速度快，适用于数据量较大的情况。但缺点需要脱机导入，无法进行增量更新，只能用于初始化数据库时使用。
2. `apoc`: 官方 `etl` 工具，支持增量更新与在线导入，但速度一般。

neo4j-admin import与apoc性能对比		
Dataset	number	
node	5900560	
relationship	18746174	
properties	30547294	
tool	time	speed
neo4j-adminimport	4m 13s 597ms.	217644 nodes/s
apoc	48m52s374ms	18822 nodes/s

我们进行了对两种工具分别进行导入的性能对比实验，结果如下表所示：从以上结果可以看出，neo4j-admin import 导入的效率约为 apoc 的 10 倍。但由于 neo4j-admin import 只支持在数据库初始化的时候进行导入，所以在初次将数据导入时我们使用 neo4j-admin import，之后的增量更新我们使用 apoc 工具。

2.4 知识图谱构建

由于，整体的数据集涵盖的分类过于庞杂，我们采用自顶向下的方法就整个数据集的与流行歌曲，歌手相关的部分进行知识图谱的构建。我们首先研究源文件，确定我们知识图谱的领域范围为流行音乐，在此基础上我们为知识图谱定义了本体与数据模式，如下图所示。以此为依据，我们从数据集中筛选数据导入 neo4j 中形成知识图谱。



STORAGE SERVER

3.1 数据库选择

本项目实际为基于知识图谱进行智能分析的项目，因此选择了以图形式存储的数据库 neo4j。neo4j 在复杂关系的存储与处理上相较于关系型数据库有着较大的优势。同时，neo4j 对于大规模数据增长与业务需求不断变化的情况有着强大的适应能力，对于日后的业务与数据的拓展极其有利。另外，neo4j 也具有非常高效的查询性能。

为了优化搜索性能与拓展搜索功能，我们选择了 elasticsearch。elasticsearch 是一个开源的分布式搜索引擎，并提供了一个分布式的实时文档存储，每个字段可以被索引与搜索。elasticsearch 使我们项目的搜索变得更加强大。

与此同时，我们还选取了高性能的 key-value 数据库 Redis 作为缓存数据库。Redis 具有以下优势：1、性能极高；2、支持数据类型丰富；3、原子性；4、支持持久化等。

3.2 数据库详情

3.2.1 NEO4J

Neo4j 存储了 5900560 个实体节点和 18746174 条关系。

3.3 查询功能实现

3.3.1 单节点及其关系查询

该功能为查询某实体的所有关系和关联对象，查询语句如下：

```
match p=(a)-[r*1..$jump]-(b)
```

```
where a:$label and b:$label and a.name='$name' and apoc.coll.duplicates(NODES(p))=[]
```

```
return p limit $num
```

该语句可以指定 4 个参数，依次可以指定要查询的关系数小于 \$jump、a 和 b 的 label、a 的 name 和返回条数。

3.3.2 两节点间多条关系查询

该功能为返回给定两实体，查询两者间存在的关系，查询语句如下：

```

match p=((a)-[r*1..$jump]-(b))
where a:$label and b:$label and a.name=' $name ' and b.name=' $name '
and apoc.coll.duplicates(NODES(p))=[]
return p limit $num

```

该语句可以指定 4 个参数，依次可以指定要查询的关系数小于 \$jump、a 和 b 的 label、a 和 b 的 name 和返回条数。

3.3.3 两节点间最短路径查询

该功能为返回给定两实体，查询两者间存在的关系，并返回最短路径，查询语句如下：

```

match p = shortestPath( (a) - [*] - (b) )
where a:$label and b:$label and a.name=' $name ' and b.name=' $name '
return p limit $num

```

该语句可以指定 3 个参数，依次可以指定 a 和 b 的 label、a 和 b 的 name 和返回条数。

最短路径我们采用了 Neo4j 内置的 shortestPath 算法进行查找。规划 Cypher 的最短路径可能会导致不同的查询计划，具体取决于需要评估的过滤条件。Neo4j 默认使用快速双向广度优先搜索算法。

3.4 查询性能优化

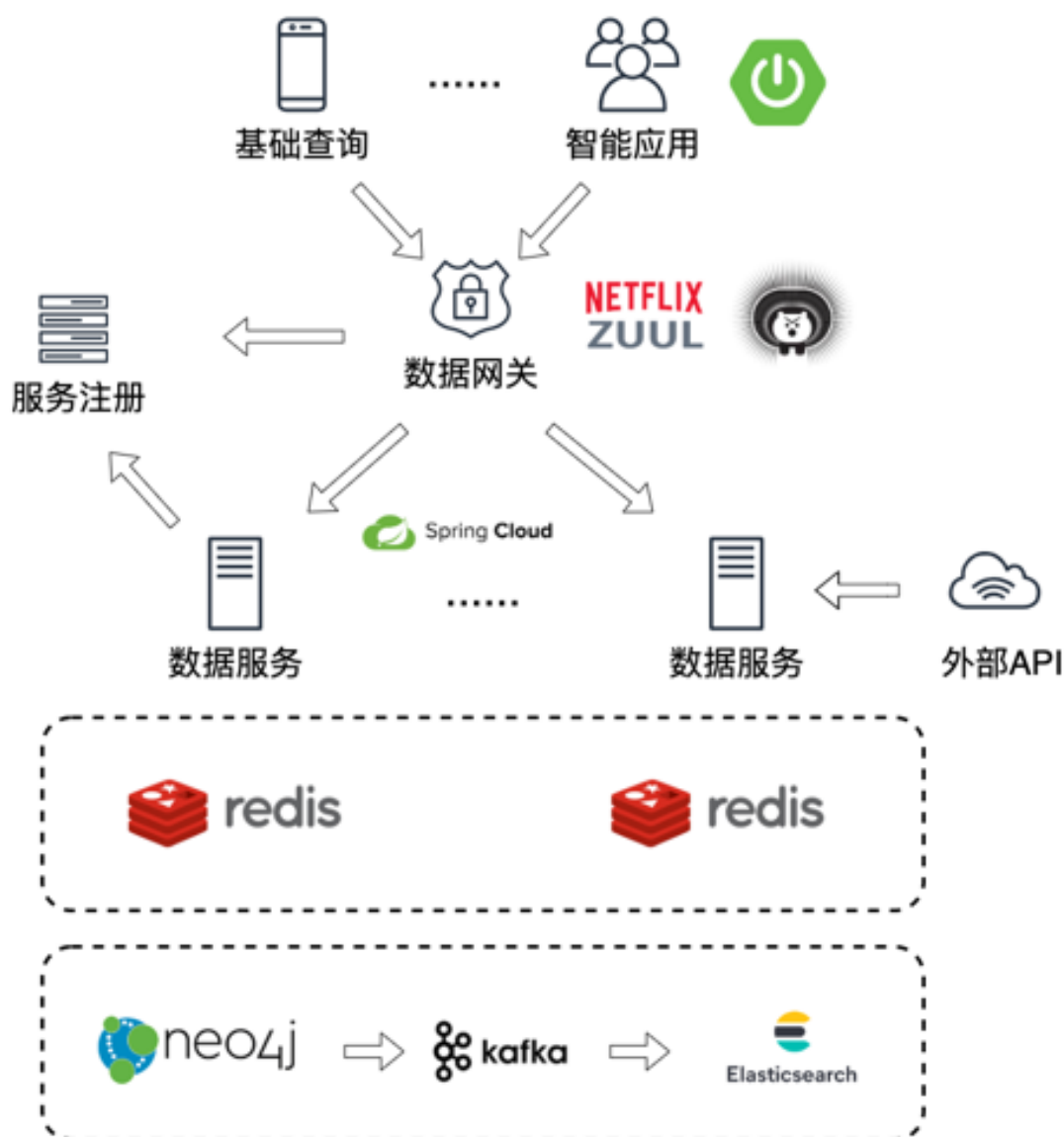
由于该数据库存储数据较多，从而影响数据库定位实体节点的性能，因此我们对于我们经常查询的属性“name”建立了索引。经实验，我们发现对于同一实体的单体查询，不建索引平均查询时间为 821ms，建立索引后平均查询时间为 337ms。（为保证实验能真实反映优化效果，我们选取位于中后部分的数据用于实验）。

为了防止查询返回环路，影响查询效率，我们调用 apoc 中的 duplicate 函数，使得带有环路的结果不被返回，减少了需要返回的数据量，降低了查询时间，也减少了不必要的返回数据。

4

BI SERVER

该模块为前端的数据可视化提供数据分析服务，为前端需要实现的针对原始数据的检索和智能应用提供数据支持与实现。



后端架构图

后端采用微服务架构，使用 Spring Cloud 实现。

每一个数据服务为一个单独的微服务，从而实现支持负载均衡的分布式系统。数据服务负责响应业务服务中所需要的数据请求，并相应地将数据库查询的结果以及调用外部 API 返回的结果封装成业务服务所需的格式。

为了实现本项目的智能分析功能，使用了以下几种外部数据 API 提供数据支撑：

1. Sematch: 计算实体与实体之间相似度与相关度。

相似度：两个实体之间的相似性，但两者之间不一定有联系。

相关度：两个实体之间的相关性，但两者之间不一定相似。

例: Apple 公司与乔布斯相关度很高, 但相似度很低。

2. Wikipedia: 提供歌手或乐队的图片数据。

请求数据格式:

```
https://en.wikipedia.org/w/api.php?action=query&titles=歌手名
&prop=pageimages&format=json
```

返回数据格式:

```
{
  batchcomplete: "",
  query: {
    pages: {
      26589: {
        pageid: 26589,
        ns: 0,
        title: "Red Hot Chili Peppers",
        thumbnail: {
          source: "https://upload.wikimedia.org/wikipedia
            /commons/thumb/1/16/
            Rhcp-live-pinkpop05.jpg/
            50px-Rhcp-live-pinkpop05.jpg",
          width: 50,
          height: 37
        },
        pageimage: "Rhcp-live-pinkpop05.jpg"
      }
    }
  }
}
```

3. tastedive: 提供数据服务以及简介

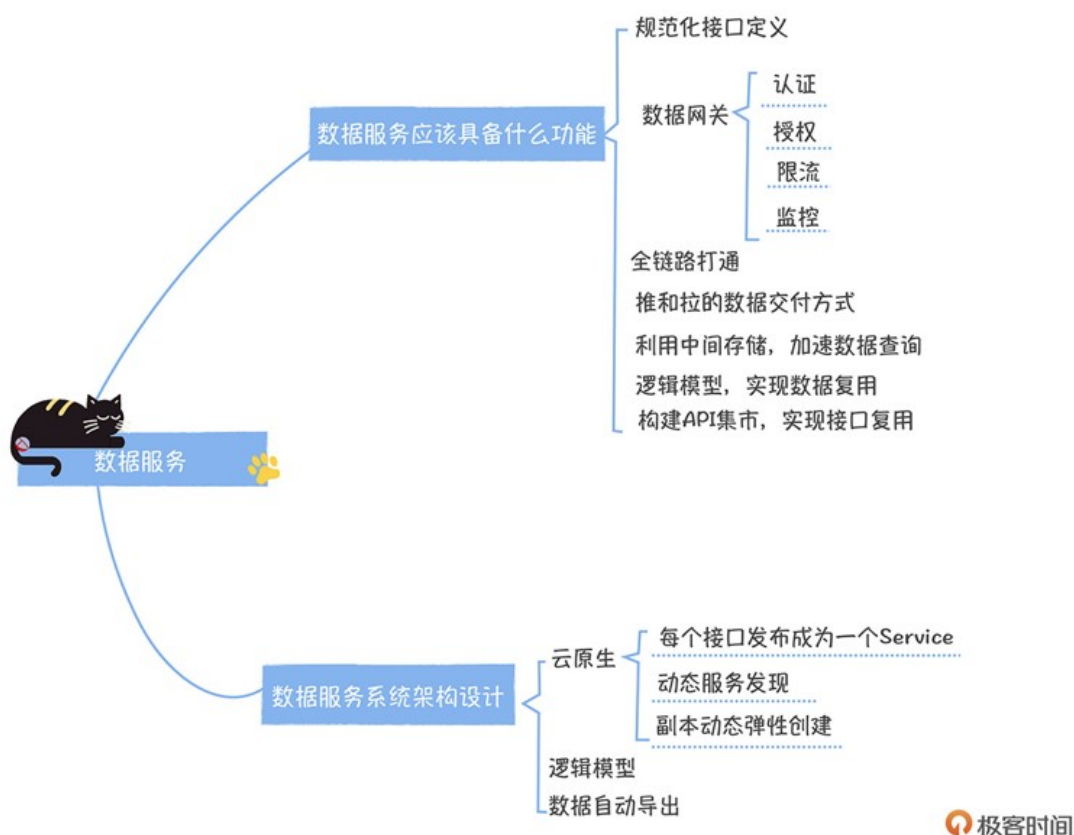
请求样例: <https://tastedive.com/api/similar?q=red+hot+chili+peppers%2C+pulp+fiction>

4. DBPedia: 提供实体属性数据

Swagger 页面:

<http://vmdbpedia.informatik.uni-leipzig.de:8080/swagger-ui.html!/Request32Types/valueRequestUsingGET>

数据服务在进行数据库和外部数据获取的时候, 会使用 Redis 作为缓存, 缓存业务服务所需要的数据, 增加下一次响应的速度。



数据中台

外部业务服务想要访问数据服务需要经过数据网关。数据网关使用 **Netflix Zuul** 实现。数据网关参照极客时间中数据网关的作用，主要负责以下几个功能：

1. 服务转发：由于每个数据服务都是单独的微服务，所以，可能分布在不同的机器上，其 IP 可能不同，且可能发生变化。所以，使用网关可以为外部访问提供统一的地址与接口，由网关向服务注册中心（**Spring Eureka** 实现）获取服务进行转发。
2. 负载均衡：因为网关向服务注册中获取到了所有可用的数据服务地址，所以可以进行负载均衡，避免单机压力过大。
3. 认证授权：由于外部请求都经过数据网关，所以可以很方便地在数据网关对所有的请求进行认证授权。本项目使用 **Spring Cloud OAuth** 和 **Spring Cloud Security** 实现，实现了用户名密码模式的 **OAuth** 验证。
4. 限流：为了避免外部恶意大量访问，可以在数据网关进行限流。本项目使用 **Zuul Ratelimit** 实现。
5. 监控：数据网关属于请求的集中点，所以对数据网关的监控可以得到所有数据服务的请求监控。本项目使用 **Netflix Hystrix** 实现。

外部数据请求可以分为基础查询和智能应用两种，基础查询直接由前端调用数据服务的接口。智能应用为独立的 **Spring Boot** 后端应用，其主要负责在 **Elasticsearch** 中进行查询，以及向数据服务请求数据，并返回给前端。

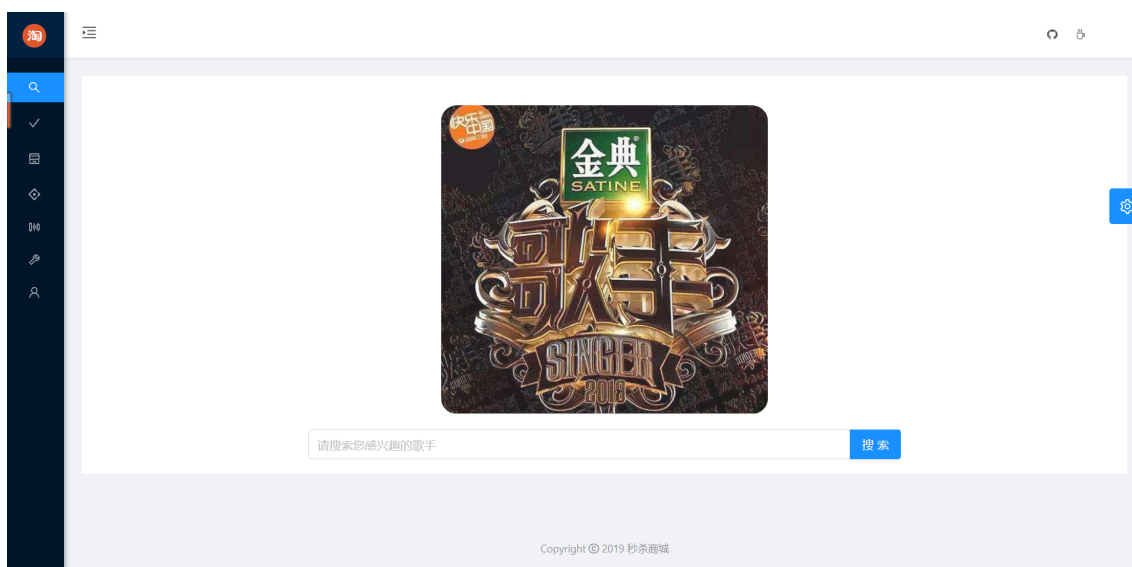
UI

5.1 智能分析

智能分析平台我们采用蚂蚁金服开发的 **Ant Design Pro** 作为框架，开发一个面向企业的歌手搜索和相似歌手推荐的商务应用平台。

5.1.1 歌手搜索界面

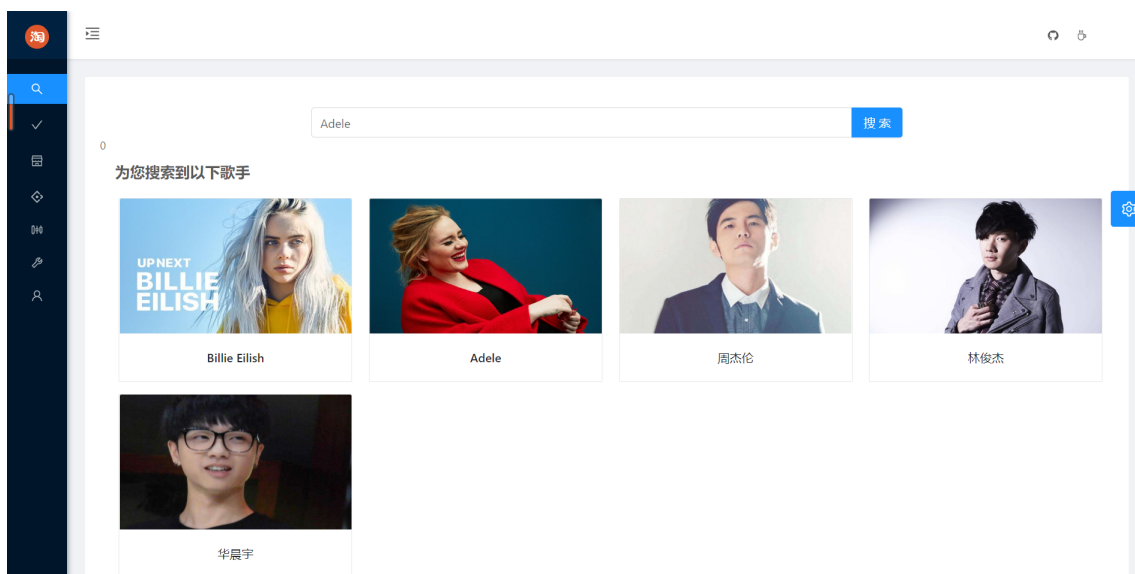
在主界面，用户可以通过歌手姓名搜索相关歌手。



歌手搜索界面

5.1.2 歌手搜索结果

根据关键词搜索出的歌手列表。




歌手搜索结果

5.1.3 歌手详情及推荐

点击歌手列表卡片可以获取歌手详情，包括基本信息、头像、音乐片段以及相关歌手推荐。

歌手 / 歌手详情

返回




Adele


个人介绍：阿黛尔·阿德金斯（Adele Adkins），1988年5月5日出生于伦敦托特纳姆，英国流行乐女歌手。

歌手音乐片段


Red Hot Chili Peppers - Californication [Official Music Video]



为您推荐的相关歌手



周杰伦

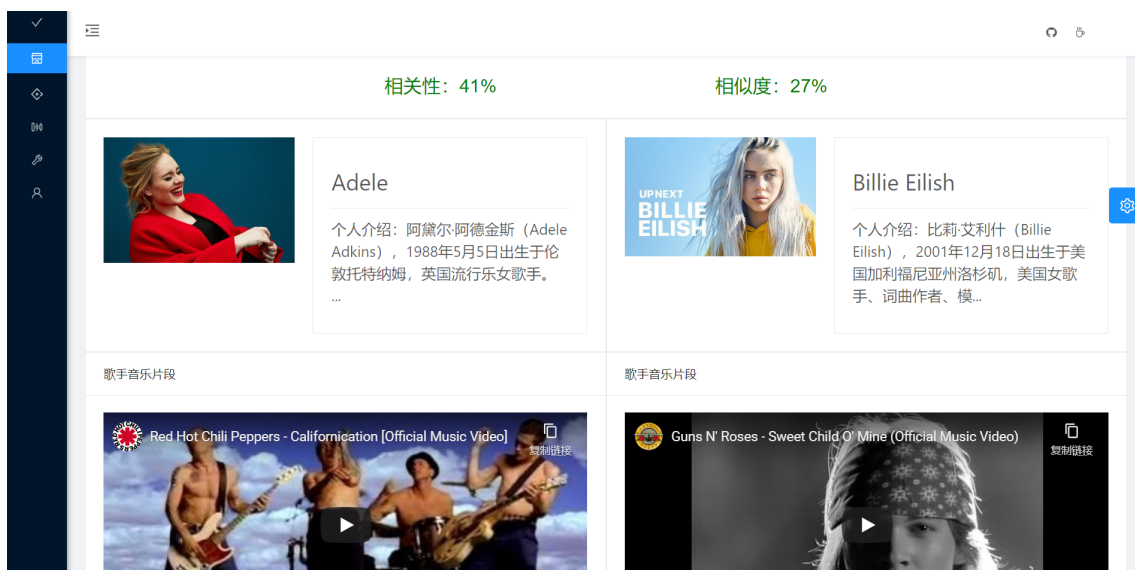


林俊杰

歌手详情及推荐

5.1.4 推荐歌手对比

点击推荐列表卡片可以获得原歌手与推荐歌手的对比矩阵，包括相似度、相关性、基本信息、头像、音乐片段以及相关歌手推荐等。



推荐歌手对比

5.2 基本查询

基本查询部分我们使用 D3.js 可视化库, 模拟 neo4j 展示界面, 搭建力导向图作为查询结果输出。

5.2.1 基本查询界面

在基本查询界面, 用户通过指定查询节点的名称、路径、返回结果数实现一个实体的查询功能; 指定开始节点和结束节点、路径、返回结果数实现两个实体的多条关系查询。

基本查询界面

5.2.2 查询结果

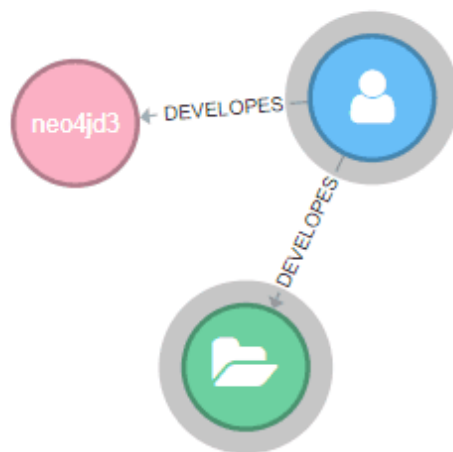
在基本查询界面, 用户通过指定查询节点的名称、路径、返回结果数实现一个实体的查询功能; 指定开始节点和结束节点、路径、返回结果数实现两个实体的多条关系查询。



查询结果

5.2.3 拓展查询

在完成基本查询之后，系统支持用户进行深入拓展查询，通过点击节点可以查询该节点在 n 跳以内的其他关联节点及其关系。



拓展查询

6

加分项总结

6.1 ETL

6.1.1 如何支持更大规模的数据？

1. 使用 Neo4j-admin import 导入数据

6.1.2 如何更好地支持数据更新？

1. 使用 APOC 进行在线更新
2. 使用 Kafka 统一 Neo4j 和 ElasticSearch 数据

6.2 STORAGE SERVER

6.2.1 如何更好地对图数据库建模？

1. 自顶向下

6.2.2 如何提高查询性能？

1. 对常用查询字段建立索引
2. 使用 apoc 的 duplicate () 过滤掉包含环路的结果

6.2.3 如何增强系统的可扩展性

1. 使用 neo4j 建立数据库，neo4j 具有设计灵活、易于拓展的特点，可根据业务灵活的设计数据模型

6.3 BI SERVER

6.3.1 如何提高查询的性能？

1. Neo4j 建立索引

2. 查询单个实体时使用 ElasticSearch
3. 使用 Redis 进行缓存

6.3.2 是否可以采用缓存？

使用 Redis 进行缓存

6.3.3 如何向通用的数据中台靠拢？

搭建具备认证、授权、限流、监控功能的数据网关。基础查询和智能应用两者作为业务服务，调用同一套通用的数据 API，获取实体-关系数据。

6.4 可视化

6.4.1 如何让用户更好地查看检索的结果？是否支持沿着图进行进一步扩展？

通过点击节点执行回调函数，在原有的 neo4j 查询结果基础上进一步拓展查询，使用户能够沿着图进行二次检索，对实体之间的关系有更清晰的了解，操作简便、结果直观。