

|  |
| --- |
| 硕 士 学 位 论 文 |

**（专业学位）**

|  |
| --- |
| **BIM数据存储研究** |

|  |  |
| --- | --- |
| **Research on BIM Data Storage** | **英文论文题目** |

|  |  |
| --- | --- |
| 院 系： | 计算机科学与技术学院 |
| 专 业： | 计算机技术 |
| 姓 名： | 张亚迪 |
| 指 导 教 师： | 王　鹏　副教授 |
| 完 成 日 期： | 2017年9月8日 |

**目录**

[摘要 IV](#_Toc494988977)

[Abstract V](#_Toc494988978)

[第1章 绪 论 1](#_Toc494988979)

[1.1 研究背景及意义 1](#_Toc494988980)

[1.2 国内外研究现状 2](#_Toc494988981)

[1.3 研究内容和本文贡献 3](#_Toc494988982)

[1.4 组织结构 4](#_Toc494988983)

[第2章 相关技术基础 5](#_Toc494988984)

[2.1 IFC基础 5](#_Toc494988985)

[2.1.1 IFC标准 5](#_Toc494988986)

[2.1.2 IFC文件格式 7](#_Toc494988987)

[2.2 全文检索技术 8](#_Toc494988988)

[2.2.1 全文检索基础理论 8](#_Toc494988989)

[2.2.2 Lucence概述 9](#_Toc494988990)

[2.2.3 ElasticSearch技术 11](#_Toc494988991)

[2.3 图数据库技术 12](#_Toc494988992)

[2.3.1 图数据与图数据库 12](#_Toc494988993)

[2.3.2 Neo4j概述 13](#_Toc494988994)

[第3章 系统总体架构 17](#_Toc494988995)

[3.1 总体架构设计 17](#_Toc494988996)

[3.2 子模块设计要点 18](#_Toc494988997)

[第4章 ElasticSearch存储关键技术研究 19](#_Toc494988998)

[4.1 IFC实体的属性数据模型概述 19](#_Toc494988999)

[4.1.1 属性 19](#_Toc494989000)

[4.1.2 属性集 21](#_Toc494989001)

[4.1.3 属性集与IFC实体关联 23](#_Toc494989002)

[4.2 半结构化存储算法 23](#_Toc494989003)

[4.2.1 冗余量估算模型 24](#_Toc494989004)

[4.2.2 生成JSON数据 25](#_Toc494989005)

[第5章 Neo4j存储关键技术研究 29](#_Toc494989006)

[5.1 IFC实体的空间关系数据模型概述 29](#_Toc494989007)

[5.2 动态建立索引算法 31](#_Toc494989008)

[5.3 基于BloomFilter的特定路径查询算法 33](#_Toc494989009)

[5.3.1 特定路径查询场景及问题 33](#_Toc494989010)

[5.3.2 BloomFilter原理 35](#_Toc494989011)

[5.3.3 基于BloomFilter的查询 37](#_Toc494989012)

[5.3.4 构建BloomFilter 39](#_Toc494989013)

[第6章 系统实验 40](#_Toc494989014)

[6.1 属性数据相关测试 40](#_Toc494989015)

[6.1.1 实验环境 40](#_Toc494989016)

[6.1.2 存储算法实验 40](#_Toc494989017)

[6.1.3 系统可用性测试 42](#_Toc494989018)

[6.2 空间关系数据相关测试 42](#_Toc494989019)

[6.2.1 实验环境 42](#_Toc494989020)

[6.2.2 动态建立索引算法实验 43](#_Toc494989021)

[6.2.3 基于BloomFilter的特定路径查询实验 43](#_Toc494989022)

[6.2.4 系统可用性测试 44](#_Toc494989023)

[第7章 总结与展望 45](#_Toc494989024)

[7.1 总结 45](#_Toc494989025)

[7.2 展望 45](#_Toc494989026)

[参考文献 47](#_Toc494989027)

# 摘要

BIM系统是一种全新的信息化管理系统，目前正越来越多应用于建筑行业中。BIM全称为Building Information Model，即建造信息模型。BIM系统要求将建筑全生命周期中的所有信息整合在统一的数据库中。然而，随着时代的发展，智慧城市概念的提出，BIM数据有了长足的增长，以往的BIM数据库存储和查询手段已经渐渐越来越不能应付数据的长足提升。所以为海量BIM数据提供一种高效快速的存储和查询手段是有必要和有意义的。

结合国内外BIM数据库研究的发展方向，可以看到，BIM数据库研究的目标是支持大数据量的存储和查询、支持更为灵活的查询方式。

基于此目标，本文选择了开源弹性搜索框架ElasticSearch并展开了研究，通过增加冗余的方式，满足用户对于属性数据的高效查询，避免了多表连接的问题。在试验中，着重解决了IFC文件解析后的数据到JSON数据的映射问题，和ElasticSearch的索引优化等问题，最终实现了IFC实体的属性数据的高效存储和查询。

其次，本文还深入研究了IFC数据标准，提出了IFC实体空间关系数据模型的存储设计。本文基于开源的Neo4j图数据库实现了IFC实体空间关系数据模型的存储，从而除了属性值查询之外，为用户提供了另外一种维度的查询方式，即空间关系的查询方式。

除此之外，本文进一步关注了一种特定的路径查询，并针对这种查询提出了基于BloomFilter的解决方案，使得查询效率得到了提升，尤其是在空值查询时能够提前失败，给用户响应。

**关键词**：建筑信息模型（BIM），IFC标准，ElasticSearch，Neo4j，BloomFilter

# Abstract

BIM system is a new information management system, and now more and more used in the construction industry. It is called Building Information Model. The BIM system requires that all information in the entire lifecycle of the building be integrated into a unified database.

However, with the development of the times, the wisdom of the city concept, BIM data has grown by leaps and bounds, the previous BIM database storage and query means has gradually become increasingly unable to cope with the data greatly improved. So it is necessary and meaningful to provide a fast and efficient means of storage and query for mass BIM data.

Combining the development direction of BIM database research both at home and abroad, it can be seen that the goal of BIM database research is to support the storage and query of large data volume and support more flexible query mode.

Based on this goal, this paper chooses the open resilience search framework ElasticSearch and studies it. It can solve the problem of multi-table join and satisfy the efficient query of attribute data by increasing the redundancy. In the experiment, we focus on solving the problem of mapping parsed IFC data to JSON data and ElasticSearch index optimization, and finally realize the efficient storage and querying of attribute data of IFC entity.

Secondly, this paper also studies the IFC data standard in depth, and puts forward the storage design of IFC entity spatial relation data model. Based on the open source Neo4j graph database, this paper realizes the storage of the IFC entity spatial relation data model, and provides the query method of the other dimension, that is, the query method of the spatial relation, other than the attribute value query.

In addition, this paper further focuses on a specific path query, and proposed a solution based on BloomFilter for this query, so that the query efficiency has been improved, especially when it is null value query it can fail in advance, and response to the user.

Keywords：BIM，IFC，ElastciSearch，Neo4j，BloomFilter

# 绪 论

## 研究背景及意义

互联网和信息技术正在变革建筑业的未来。近年来，BIM（Building Information Model，即建筑信息模型）技术在国内外建筑行业得到广泛和应用。BIM技术以建筑工程项目的各项相关信息数据作为基础，建立起三维的建筑模型，通过数字信息仿真模拟建筑物所具有的真实信息。基于BIM技术建立的系统为建筑的全生命周期管理提供平台，参建各方利用统一的建筑信息模型，在设计、施工、项目管理、项目运营等各个过程中，实现项目协同管理，减少错误、节约成本、提高质量和效益。BIM技术开启建筑信息化第二次革命，让建筑信息化从CAD时代迈入BIM时代。

BIM技术的核心是建筑全生命周期过程中的BIM数据存储和利用。BIM数据库系统建立完整的建筑信息模型数据库，模型包含建筑所有构件、设备等几何和非几何信息以及之间关系的信息，模型信息随着建设阶段，不断深化和增加。同时，BIM数据库实现了建筑信息模型数据的开放利用，基于BIM数据库，建筑信息提供者、项目管理者与用户之间可以实时、方便地进行信息交互。

信息的标准化表达是实现BIM数据库的基础。IFC（Industry Foundation Class）标准，作为国际通用的数据表示和交换标准，定义了建筑工程项目全生命周期的数据格式，内容丰富、涉及领域广，成为BIM技术的主流数据交换标准。目前，国内外BIM数据库的研究主要基于IFC标准展开。

随着智慧城市的提出、建筑体量大型化趋势的到来，建筑数据有了长足的增长。2015年11月6日完工的广州周大福金融中心工程总高度530米，占地面积2.6万平方米，总建筑面积50.77万平方米，产生项目图纸5万多张，图纸变更3000余次 [1]。如此大体量的建筑数据为BIM数据库的设计提出了更高的要求。解决如何高效、快速地存储和管理海量BIM数据的问题迫在眉睫。

当前，BIM数据库按数据库类型可分为关系型BIM数据库、面向对象型BIM数据库、关系-对象型BIM数据库和NoSql型数据库【1】。尤其是进入大数据时代以来，分布式技术如Hbase、Redis等被越来越多地应用在BIM数据库研究中，提高了BIM数据库的性能。然而，海量BIM数据的存储设计方面依然存在一些局限，如何结合IFC标准数据特点进行数据存储设计成为研究的重点。

## 国内外研究现状

1. **关系型BIM数据库**

关系型BIM数据库,其主要采用关系型数据库存储BIM数据。一般根据IFC标准的逻辑大纲设计数据逻辑模式。譬如针对IFC中每个实体定义一个表，并根据该实体的每个属性定义一个表的字段。现有的关系型BIM数据库包括VTT Building and Transport And SECOM Co．开发的一个基于SQL Server的IFC数据库[1],清华大学张洋博士开发的“BIM信息集成平台”[2]，Cruz等开发的ACTIVe3D系统采用Oracle进行IFC数据存储[3][4][5]。由于IFC实体的具有大量可选属性，很多属性值为空(即稀疏性)，但关系数据库仍为其预留大量空间，从而导致存储空间的浪费。另外，关系数据库的扩展性较差，难以支持海量BIM数据的存储，所以关系型数据库已经不能满足BIM数据管理的需要。

1. **面向对象型BIM数据库**

鉴于IFC模型的面向对象特性，一些学者提出应用面向对象数据库存储BIM数据。Faraj利用面向对象数据库ObjectStore存储IFC数据[6]。陆宁设计和实现了基于面向对象数据库Versant Object Database 8的IFC数据库，并通过与关系数据库对比，指出面向对象型IFC数据库效率更高[7]。但鉴于面向对象数据库本身理论不完善、价格昂贵等问题，存在难以推广的问题，且其可扩展性较NoSql数据库差。

1. **对象关系型BIM数据库**

面向对象技术与传统关系型数据库技术相结合而形成的数据库系统，也可以说是一种扩展关系数据库，使它具有一定面向对象数据库特征。Kang等利用对象关系数据库Cubrid存储IFC数据[8]。但目前对象关系数据库尚不成熟，没有广泛使用的商业软件。

1. **键值型BIM数据库**

当关系型数据库在可扩展方面几乎已经达到极限时，键值对数据库开始流行起来，它为解决大数据的需求提供了解决方案。这是一种NoSQL（非关系型数据库）模型，其数据按照键值对的形式进行组织、索引和存储。

开源的项目BIMServer采用键值对数据库Berkeley DB来进行数据的存储。Berkeley DB是一个开源的嵌入式数据库系统，提供了一系列直接访问数据库的函数，支持存储任意类型的键/值对，具有简单、小巧、可靠、高性能的特点。BIMServer赋予每一个IFC实体一个全局唯一的ID作为key，将IFC实体的所有字段值解析为字节数组作为value，将key/value对存储与Berkeley DB中，能够支持基于IFC实体ID的快速查询和应用。其局限在于键值对存储IFC实体信息的方式比较单一，不能够支持任何关系数据的存储，也不能支持基于字段的查询。

清华大学的刘强等为实现IFC模型在Web应用中的应用，使用分布式键值对存储系统Redis建立了IFC数据的高速缓存[9]。它将IFC文件以数据流方式写入Redis缓存，利用IFC文件的行编号进行索引，大幅提升了IFC文件的解析效率。同时结合内存中的EXPRESS字典使用JSON格式构建IFC对象模型，从而支持面向Web的IFC模型对象的传输和查询等应用。这种方式的局限类似于Berkeley DB，它只能支持基于IFC实体ID的查询。

1. **列式数据库**

清华大学的余芳强等基于分布式的开源列式数据库Hbase来解决基于列值的查询问题和关系数据join问题。不同于关系数据库的按行存储数据的方式，Hbase以列为单位存储数据，使用列族（Family）和列名共同定义一个列，擅长以列为单位存取数据。Hbase支持动态添加列，从而支持半结构化的BIM模型设计，以一定的冗余避免了多表连接操作，极大地提高了信息提取速度。

但是其半结构化的数据库设计只关注了IFC类型的可交换实体和非可交换实体，存在一定的局限。

## 研究内容和本文贡献

以往的BIM关系数据库已经不能满足海量BIM数据存储和管理的需要，NoSql类型的BIM数据库，基于分布式技术、半结构化的存储设计一定程度上解决了海量BIM数据查询时的效率不高的问题，但是依旧存在一定的局限性。在IFC标准中为建筑构件定义了丰富的属性信息和空间关系信息，如窗的高度、墙的面积、窗与墙的关系等，在建筑初步设计、设计变更、施工图设计、施工实施等阶段都具有重要的应用意义。现有的BIM数据库设计只能采用多表连接的方式来获取属性数据和空间关系数据，效率十分低下。

本文在深入研究IFC标准中属性数据的基础上，结合ElasticSearch弹性搜索技术，提出了属性数据半结构化存储算法，同时为了减少冗余，本文基于对属性数据的估算引入冗余控制参数α，提出动态地合并属性数据到IFC实体数据的策略，从而在实现高效提取属性数据的同时，有效地控制了冗余。

其次，本文深入研究IFC标准中空间关系数据，结合Neo4j图数据库技术，实现IFC对象空间关系数据模型的存储。同时，提出了基于用户查询集的动态建立索引的算法，解决了单机存储空间存在限制时控制索引大小问题。

除此之外，本文进一步关注了一种特定的路径查询，并针对这种查询提出了基于BloomFilter的查询实现算法，相比于原生的Neo4j cypher查询，效率得到了明显提升。尤其是在空值查询时能够提前失败，给用户响应。

## 组织结构

本文的组织结构大致如下：

**第1章**，描述了本文的研究背景与研究意义。

**第2章**，为本文的相关技术基础与背景知识给出了一个大致的介绍。主要涉及到IFC相关知识、全文检索Lucene和弹性搜索框架ElasticSearch、图数据和Neo4j图数据库等。

**第3章**，介绍了系统的总体架构。该系统结合ElasticSearch搜索框架和Neo4j图数据库，从IFC文件解析、数据模式映射、数据存储和查询等模块进行综述。

**第4章**，介绍了本文如何利用ElasticSearch处理和存储IFC实体的属性信息。以数据处理流程的先后顺序展开介绍，主要涉及到IFC属性和属性集的详细内容、IFC文件解析、JSON数据映射的定义和数据生成、索引过程、查询技术等部分。

**第5章**，介绍了本文如何利用Neo4j处理和存储IFC实体空间关系数据。以数据处理流程的先后顺序展开介绍，主要涉及到IFC空间关系数据模型的详细内容、图数据模型、存储技术、查询技术以及基于BloomFilter的查询优化等。

**第6章**，对本文提出的方案进行了实验验证，证明了本文方法的可行性和有效性。

# 相关技术基础

## IFC基础

### IFC标准

IFC是由buildingSMART以工业的产品资料交换标准STEP编号ISO-10303-11的产品模型信息描述,用EXPERSS语言为基础，基于BIM中AEC/FM相关领域信息交流所指定的资料标准格式。有专家认为IFC如同网络通信标准HTML一样，IFC不属于任何BIM软件专有，而加入IFC标准认证的各领域及不同软件也日益增加。许多公司或教育单位也加入研究并开发相应的应用，同时提供免费试用源代码，以此吸引更多人参与IFC的研究与发展。基于BIM的IFC标准已经发展10年有余，渐渐受到学术界与业界重视，IFC不断发展会是AEC相关信息交换的重要标准。

IFC标准同样也是一个类似面向对象的建筑数据模型。IFC 模型包括建筑整个生命周期内的各方面的信息，其中包含的信息量非常大而且涵盖面很广。IFC标准的目的是支持用于建筑的设计、施工和运行的各种特定的软件的协同工作。正因为如此，IFC标准是目前对建筑物信息描述最全面、最详细的规范。这证明了IFC 模型是建筑工业和设备制造工业之间的数据模型交换的最好方法。

为此，IFC标准的开发人员充分地应用了面向对象分析和设计方法，并设计了一个总体框架和若干原则将这些信息包容进来并加以很好地组织，这就形成了IFC的整体框架。IFC的总体框架是分层和模块化的，整体可分为四个层次，从下到上依次为资源层、核心层、共享层、领域层。每个层次内又包含若干模块，每个模块内又包含了不少信息。图2-1是IFC模型总体结构图。

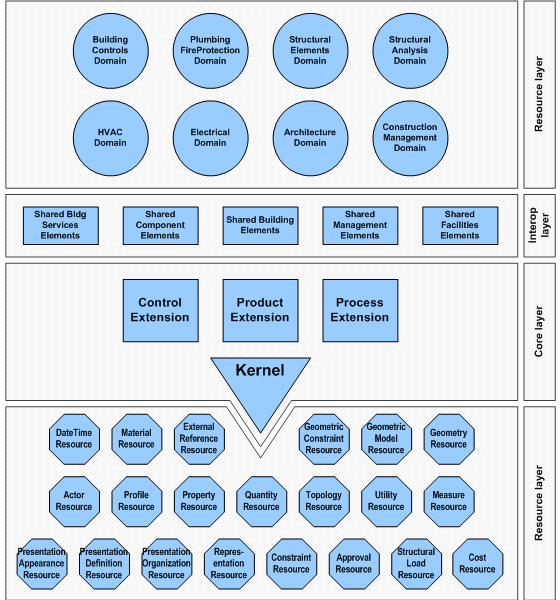


图2-1 IFC模型总体结构图

其中：

1. 信息资源层（Resource layer）描述标准中用到的基本信息，如几何、尺寸、材料等基本元素信息，是整个信息模型的基础。这些信息可与其上层（核心层、共享层和领域层）的实体连接，用于定义上层实体的特性。
2. 核心层（Core Layer）定义了建筑工程信息的整体框架。这个层次提炼了一些适用于整个建筑行业的抽象概念，不仅包括建筑对象的位置和几何形状等，同时也定义了建筑对象之间的关系，其可以反映现实世界的结构。
3. 共享层（Interoperability layer）分类定义了一些适用于建筑项目各领域（如建筑设计、施工管理、设备管理等）的通用概念，以实现不同领域间的信息交换。
4. 领域层（Domain Layer）：定义了一个建筑项目不同领域（如建筑、结构、暖通、设备管理等）特有的概念和信息实体，形成领域内的专题信息。

### IFC文件格式

IFC标准采用了EXPRESS语言作为描述语言，来描述IFC模型信息。一个完整的IFC模型包括类型定义、函数、规则、及预定义属性集组成。其中，类型定义是IFC模型的主要组成部分。类型定义分为定义类型（Defined Types）、枚举类型（Enumeration）、选择类型（Select Types）和实体类型（Entities）。IFC标准的当前版本（IFC 2X3）中包含了117个定义类型、164个枚举类型、46个选择类型和653个实体类型。实体采用面向对象的方式构建，比如门、窗，都是一个实体的实例。而定义类型、枚举类型、选择类型通常作为属性值出现在实体的实例中。下面图2-2中展示的是IfcDoor实体类型的EXPRESS定义。

图2-2

首行“Entity”表示是实体类型的定义。第二行表示它是IfcBuildingElement类型的子类。由于实体是采用面向对象的方式构建的，当然具有继承的特性。第三行和第四行则定义了两个属性OverallHeight（高度）和OverallWidth（宽度）。“OPTIONAL”表示这个属性是可选的，“IfcPositiveLengthMeasure”是一个定义类型（Defined type），它用来表示OverallHeight（高度）和OverallWidth（宽度）这个属性的值是大于0的长度测量值。但是IfcDoor的属性绝不是只有这两个，它也继承了IfcBuildingElement的属性，同时IfcBuildingElemnt也继承了更上一层父类的属性。所以，IfcDoor的属性是自身的定义的属性与所有父类的属性的集合，且严格按照顺序。

那么一个IFC实体的实例的信息是如何在文件中存储的呢？这就要引出IFC SPF文件。

IFC SPF(STEP Physical File)文件是IFC标准所使用的主要数据交换文件。其内容紧凑，无冗余信息，大大地减少了数据存储占用空间。一个完整的IFC SPF文件包含两个部分：

1. 标头段，包含一些文件信息。
2. 数据段，包含IFC整个模型的信息。

图2-3



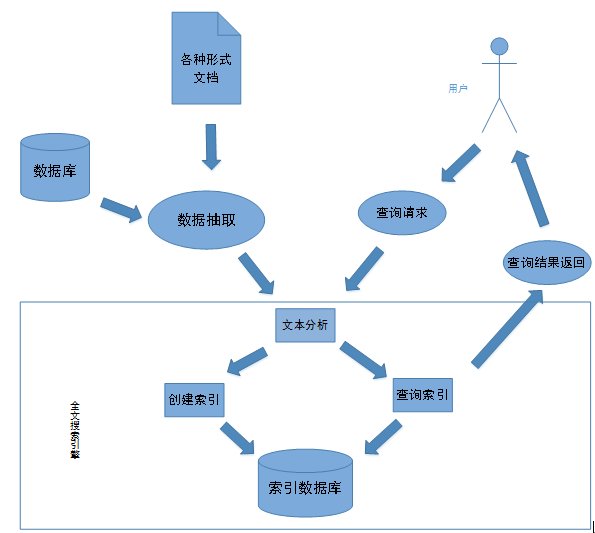
图2-3中的解释是：

1. 行1是文件开始语句，表示ISO发布的STEP标准。对应文件最后一行（第92行）的END-ISO-10303-21。
2. 行2-6，表示文件信息，以HEADER关键字开头，以ENDSEC关键字结束。包括文件描述、文件名称、文件使用的IFC标准。
3. 行7-92是数据段。以DATA关键字开头，以ENDSEC关键字结尾。中间的每一行表示一个IFC实体的实例数据。#后面的数字是一个唯一的编号，可以不连续，而且不一定是增序，标识一个IFC实体的实例。等号右端首先是IFC实体的类型名，括号内是按照EXPRESS定义的包括继承属性的所有属性的属性值，值的类型可能是定义类型、选择类型、枚举类型。

## 全文检索技术

### 全文检索基础理论

全文检索技术，就是以数据诸如文字，声音，图像等为主要内容，以检索文献资料的内容而不是外表特征的一种检索技术，全文检索是信息检索技术的一种，主要是把用户的查询请求和全文中的每一个词进行比较，不考虑查询请求与文本语义上的匹配。在信息检索工具中，全文检索是最具通用性和实用性的。

图2-4全文搜索一般流程

全文搜索的一般流程如图2-4所示。首先，从各类文档和数据库当中抽取数据，经过文本分析以后，将多种异构数据转化成同构的结构化或者半结构化数据，创建全文索引，建立索引数据库。目前应用最广的全文索引就是以倒排索引为核心的一系列的技术。当索引数据库建立完全以后，人们就可以通过查询接口查询文档。

由图2-4中可以看出，全文检索系统中最核心、最关键的部分是全文检索引擎部分，这部分从功能模块上可以划分为文本分析模块、创建索引模块、查询索引模块。索引的准备工作和搜索的应用都是建立在这个引擎之上。由此可见，一个全文检索应用的优异程度，根本上是由全文检索引擎来决定。因此提升全文检索引擎的效率即是我们提升全文检索应用效率的根本。目前应用最广泛的数据库就是Lucence以及其一系的ElasticSearch与Solr。

### Lucence概述

Lucene是非常优秀的成熟的开源的免费的纯Java语言的全文索引检索工具包。Lucene是一个高性能、可伸缩的信息搜索库（Information Retrieval Library）。它使你可以为你的应用程序添加索引和搜索能力。Lucene的作者Doug Cutting是资深的全文索引/检索专家。他最开始将此项目发布在他本人的主页上，2001年10月贡献给APACHE，成为APACHE基金的一个子项目。

Lucene作为一个全文检索引擎，其具有如下突出的优点：

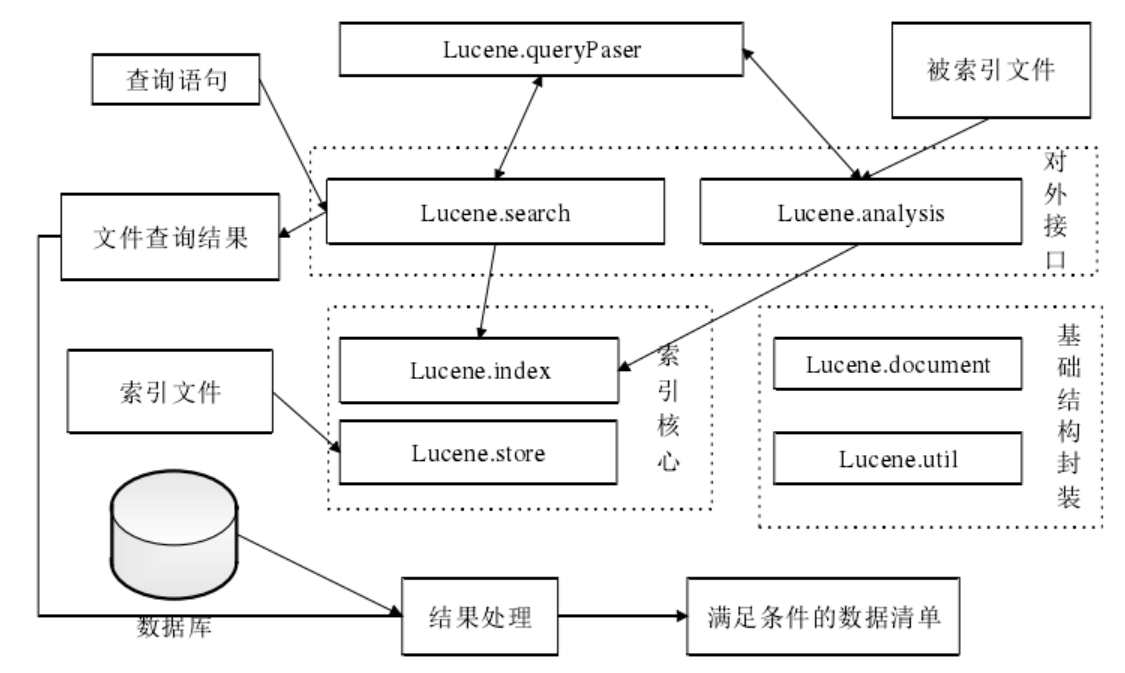
1. 索引文件格式独立于应用平台。Lucene定义了一套以8位字节为基础的索引文件格式，使得兼容系统或者不同平台的应用能够共享建立的索引文件。
2. 在传统全文检索引擎的倒排索引的基础上，实现了分块索引，能够针对新的文件建立小文件索引，提升索引速度。然后通过与原有索引的合并，达到优化的目的。
3. 优秀的面向对象的系统架构，使得对于Lucene扩展的学习难度降低，方便扩充新功能。
4. 设计了独立于语言和文件格式的文本分析接口，索引器通过接受Token流完成索引文件的创立，用户扩展新的语言和文件格式，只需要实现文本分析的接口。
5. 已经默认实现了一套强大的查询引擎，用户无需自己编写代码即使系统可获得强大的查询能力，Lucene的查询实现中默认实现了布尔操作、模糊查询（Fuzzy Search）、分组查询等等。

图2-5 lucence代码结构

Lucene的代码结构大致如图2-5所示，主要分为索引建立，与文档查询两个主要流程。首先，在索引建立的过程中，先将需要被索引的文件通过分析器分析好以后建立好倒排索引，放入索引核心的索引数据库当中。而当用户发起一个查询流程的时候，通过查询，将通过查询器，查询索引结果，通过查询语句的相应逻辑给出查询结果。

### ElasticSearch技术

ElasticSearch 是分布式实时搜索、实时分析，实时存储引擎，简称（es)。ElasticSearch使用Java开发并使用Lucene作为其核心来实现所有索引和搜索的功能，通过简单RESTful API来隐藏Lucene的复杂性，从而让全文搜索变得简单。它的检索效率超出你的想象，从10亿的数据查出一条只需要1-2秒内。此外，ElasticSearch还具有支持实时分析 、实时搜索 、可分布、可扩展到上百台PB机器的特性。

下面主要来介绍一些ElasticSearch中的一些相关术语：

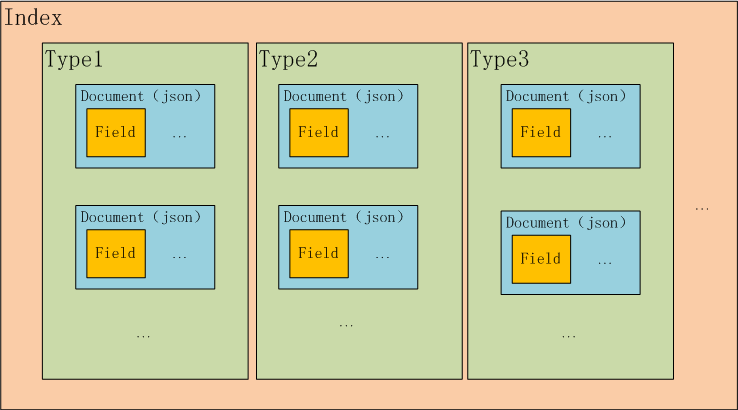
1. 索引（index）：一个索引就是一个拥有几分相似特征的文档的集合。比如说，你可以有一个客户数据的索引，一个产品目录的索引，还有一个订单数据的索引。一个索引由一个名字来标识（必须全部是小写字母），并且当我们要对这个索引中的文档进行索引、搜索、更新和删除的时候，都要使用到这个名字。
2. 类型（type）：在一个索引中，你可以定义一种或多种类型。一个类型是你的索引的一个逻辑上的分类/分区。
3. 文档（document）：一个文档是一个可被索引的基础信息单元。比如，你可以拥有某一个客户的文档，某一个产品的一个文档，当然，也可以拥有某个订单的一个文档。文档以JSON（JavaScript Object Notation）格式来表示。如图2-6所示。

图2-6 ElasticSearch存储逻辑

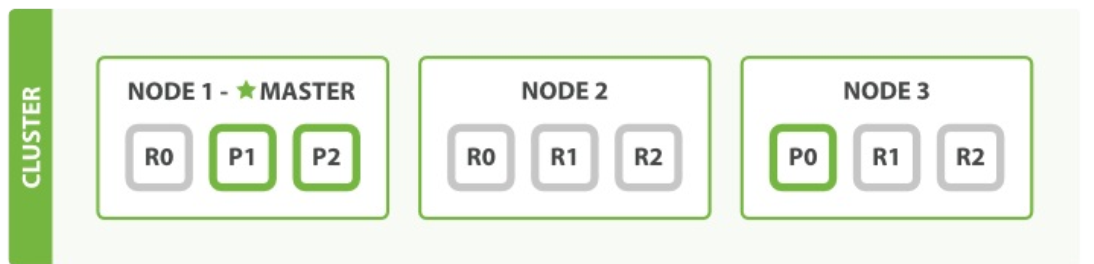
1. 分片(shard)：一个索引保存了大量的文档数据，那这些数据是如何存储的呢？其实这是把很多数据，分布的放在每个分片中，分片又被放到集群中的各个机器上。每个分片都独立的“索引”（即可以增加、删除，修改、查询）。这样做的好处有两点:1.横向扩展，水平分割数据容量;2.可以在分片上并行的进行操作。如图5所示，每个索引的分片均匀的分布到了整个集群当中，同时每个分片还有一个副本，保证数据的高可用性。

图2-7 ElasticSearch集群逻辑图

另外，ElasticSearch支持REST API对索引进行CRUD（增删改查）操作。curl工具是一种可以在命令行访问url的工具，支持get和post请求方式。-X指定http请求的方法，-d指定要传输的数据。要使用curl访问ElasticSearch索引的数据，其语法格式如图2-8：

图2-8

其中，<REST Verb>是REST风格的语法谓词；<Node>指定集群中的某个节点的IP；<port>指定节点端口号，默认9200；<Index>指定索引名；<Type>指定文档类型；<ID>指定文档的ID。

所以ElasticSearch为Lucene提供了一种高效的分布式数据存储方案，提供了高效的数据吞吐能力，以及处理大数据的能力。同时REST API使得整个集群更加易用，为开发者提供了快速开发应用的条件。

## 图数据库技术

### 图数据与图数据库

图数据库源起欧拉和图理论，也可称为面向/基于图的数据库，对应的英文是Graph Database。图数据库的基本含义是以“图”这种数据结构存储和查询数据，而不是存储图片的数据库。它的数据模型主要是以节点和关系（边）来体现，也可处理键值对。它的优点是快速解决复杂的关系问题。

图具有如下特征： 包含节点和边；节点上有属性（键值对）；边有名字和方向，并总是有一个开始节点和一个结束节点；边也可以有属性。图可以说是顶点和边的集合，或者说更简单一点儿，图就是一些节点和关联这些节点的联系（relationship）的集合。图将实体表现为节点，实体与其他实体连接的方式表现为联系。

图将实体表现为节点，实体与其他实体连接的方式表现为联系。我们可以用这个通用的、富有表现力的结构来建模各种场景，从宇宙火箭的建造到道路系统，从食物的供应链及原产地追踪到人们的病历，甚至更多其他的场景。

通常，在图计算中，基本的数据结构表达就是：

图数据库存储一些顶点、边与表中的数据。它们用最有效的方法来寻找数据项之间、模式之间的关系，或多个数据项之间的相互作用。

一张图里数据记录在节点，或包括的属性里面。最简单的图是单节点的，记录了一些属性。一个节点可以从单属性开始，成长为成千上亿，虽然会有一点麻烦。从某种意义上讲，将数据用关系连接起来分布到不同节点上才是有意义的。

图计算是在实际应用中比较常见的计算类别，当数据规模大到一定程度时，如何对其进行高效计算即成为迫切需要解决的问题。大规模图数据，例如支付宝的关联图，仅好友关系已经形成超过1600亿节点、4000亿边的巨型图。要处理如此规模的图数据，传统的单机处理方式显然已经无能为力，必须采用由大规模机器集群构成的并行图数据库。

在处理图数据时，其内部存储结构往往采用邻接矩阵或邻接表的方式，在大规模并行图数据库场景下，邻接表的方式更加常用，大部分图数据库和处理框架都采用了这一存储结构。

随着数据量的不断变大，图数据库也变成了大数据时代的研究热点。

### Neo4j概述

Neo4j是基于Java开发的高性能的、高可靠的、高可扩展的NOSQL图形数据库。Neo4j支持所有的图数据变更操作放在事务中处理，确保数据的一致性。Neo4j单个服务器实例可以应对有着大量节点和关系的复杂图数据模型，处理能力可达到数十亿级别。此外，Neo4j还提供了非常快的图算法，如图遍历、最短路径等。Neo4j可以作为嵌入式的图形引擎用于各种需要快速开发的图应用当中，其轻量、高性能的优势使其越来越受到关注。

下面分别介绍Neo4j的基本概念、查询语言和索引。

1. Node、Relationship、Property、Label

在Neo4j中，有两种基本数据类型，Node（节点）和Relationship（关系）。Node通过Relationship所定义的关系连接起来。同时可以在Node和Relationship上赋予key/value形式的属性，来表示详细的信息，以便对Node和Relationship进行查询。同一类型的节点还可以被赋予Label（标签）。Neo4j会根据Label将节点组织为不同的set（集合），基于Label来进行节点的查询将使得查询不再是面向整个图，而是面向某些set。Label是可选的，一个节点可以没有Label。同时，一个节点也可以有多个Label。通过节点、关系属性和标签可以构建出一个大型的图结构，再通过一些列的图操作来进行数据的管理和应用。这就是Neo4j的图数据模型。下图是一个简单的图模型：

图2-9 简单图模型

图2-9是一个含有3个节点的图模型,两个带有“Person”标签的节点，分别有两个属性，“name”和“born”。还有一个带有“Movie”标签的节点，有两个属性“title”和“released”。“Tom Hanks”节点（我们暂且以属性值标识这个节点）与“Forrest Gump”有一个“ACTED\_IN”的关系，且关系具有属性“roles”，其语义是Tom Hanks出演了电影Forrest Gump，出演的角色是“Forrest”。另一个关系“DIRECTED”，表示“Robert Zemeckis”执导了影片“Forrest Gump”。

Neo4j提供了图遍历的traverse API，以某一个节点为起始节点，可以采用深度优先或者广度优先的遍历算法遍历整个图中的节点。

1. Cypher查询语言

Neo4j提供了Cypher查询语言。Cypher是一种声明式的图查询语言，允许对图数据进行有效的查询和更新。Cypher是一个比较简单但语言非常强大的语言。非常复杂的数据库查询可以很容易地通过Cypher来表达，这很大程度上方便了开发者或用户对Neo4j做点对点模式（ad-hoc）的查询操作和其它数据库操作。Cypher参考了SQL的结构，使用各种子句建立查询。子句连接在一起，并且彼此之间提供中间的结果集。例如，一个MATCH子句的匹配结果是下一个子句的上下文。下面是部分查询子句：

MATCH：匹配模式，这是从图形中获取数据的最常用方法。

WHERE：过滤条件。

RETURN：返回所需要的。

基于图2-9的模型，如果要查找Tom Hanks演过的电影，Cypher语句可以写为如图2-10形式：

图2-10

需要注意的是，Cypher并不是一个静态的语言，不断发展的新版本会加入一些新的功能，有时候还会删除一些旧的功能。例如在Cypher 2.2之前的版本中，START子句被用来查找指定的ID的节点作为起始节点，但是在2.2及以后的版本中，START被完全禁止了，取而代之的是用MATCH子句来查找起始节点。每一个版本的Neo4j支持某些版本的Cypher，如Neo4j 3.2支持的Cypher版本有3.2、3.1、2.3。当升级Neo4j版本时，需要注意查看相应的Cypher版本是否支持。

1. Legacy Index和Schema Index

Neo4j提供了属性上的索引，使得用户可以通过属性值快速找到节点。索引包含Legacy Index 和Schema Index两种类型，它们都是基于Lucene实现的在Neo4j 2.0版本之前的索引被称为Legacy Index。Legacy Index能够提供全文检索的能力，而在Schema Index当中没有。这也是Neo4j 2.0版本及以后的版本还保留着Legacy Index的原因之一。使用Legacy Index往往需要一个“起始节点”，使用Cypher语句时需要借助START子句来调用。

在Neo4j 2.0 及以后的版本中Legacy Index不再是官方推荐的索引，而是推荐使用Schema Index。Schema Index需要基于Label和Property使用。它进行了一些优化，使得MATCH子句中查询节点变得更快。

创建Schema Index的Cypher语法如图2-11：

图2-11

其中的Label和Property分别对应具体的标签和属性。

# 系统总体架构

本文在深入研究了IFC标准的基础上，提出基于ElasticSearch弹性搜索框架来实现IFC数据存储，并针对属性数据的特点实现半结构化存储，以解决属性数据查询时的join效率低下的问题。同时，本文基于Neo4j图数据库来实现IFC实体空间关系信息的存储。

本章将介绍系统的总体架构及各个模块的设计。

## 系统总体架构

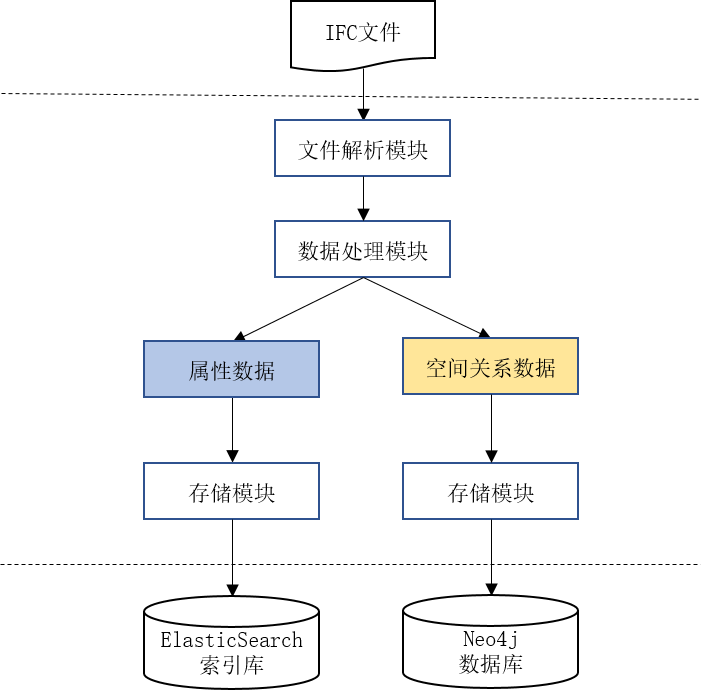
按照数据处理过程，该系统的模块可以分为：IFC文件解析模块、数据处理模块、数据索引等模块。结合ElasticSearch搜索框架和Neo4j图数据库，其总体架构如下图3-1：

图3-1 系统总体架构

系统中各个模块的执行流程是：

1. 使用IFC文件作为数据来源，读取IFC文件到系统。
2. 文件解析模块将解析IFC文件为内存中的IFC对象数据集合。
3. 数据处理模块将对IFC对象数据进行处理，其中的重点是按照一定的策略生成属性数据和空间关系数据。
4. 存储模块将处理得到的属性数据索引到集群中的索引库中去，将处理得到的空间关系数据存储到Neo4j数据库中。

## 子模块设计

### 文件解析模块

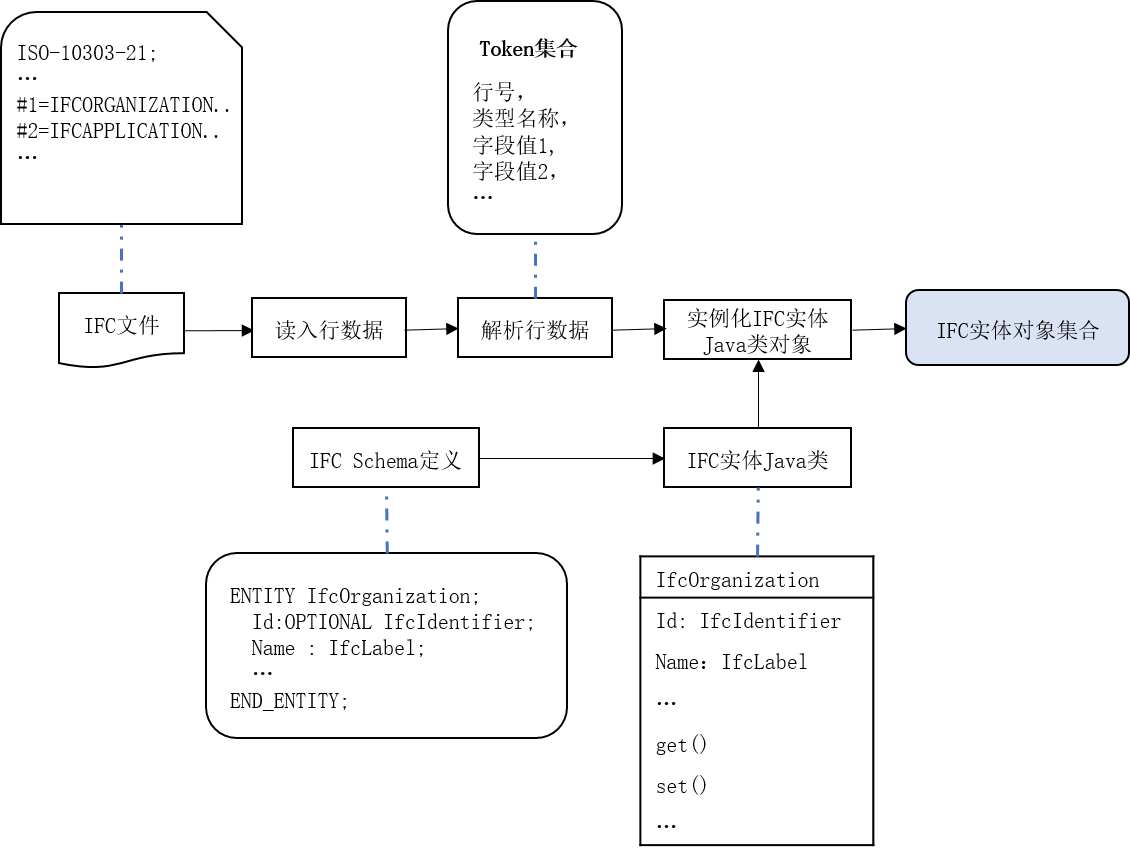
IFC文件解析模块的主要目的是解析IFC文件，得到Java类对象的集合。解析过程如下：逐行读入IFC文件的每条数据字符串，解析获得token集合，包括行号、类型名称、多个字段值等。行号是IFC对象的ID，类型名称是IFC对象的类型，字段值对应了IFC Schema中对IFC实体类型的Express定义。基于IFC schema的Express定义语言，构建IFC实体类型、定义类型、枚举类型、选择类型对应的Java类。然后结合Token集合中的IFC实体的数据实例化相应的IFC类对象。所有行数据解析完毕，最终得到IFC实体对象集合。解析流程图如图3-2所示。

图3-2 IFC文件解析流程

### 数据处理模块

数据处理模块的主要目的是将IFC实体对象的数据按照一定的策略转化为对应的数据格式，为存储做数据准备。在此模块，重点针对IFC实体的属性数据和空间关系数据进行处理。本文提出了属性数据的半结构化存储方式，将属性数据合并到IFC实体对象的数据中，以避免提取属性数据时join产生的效率低下的问题。同时提出了基于Neo4j的IFC实体的空间关系数据存储方式。下面针对属性数据、空间关系数据的处理进行分别介绍。

1. 属性数据处理

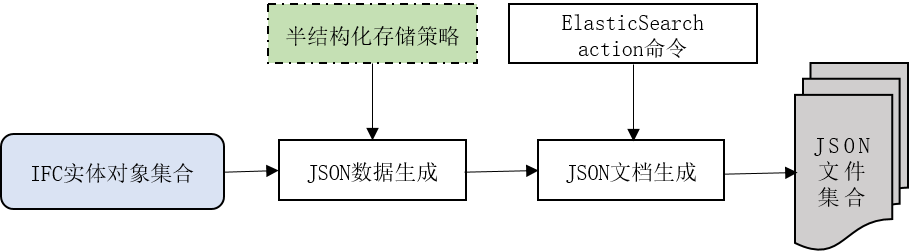
ElasticSearch支持的数据类型是JSON数据。属性数据处理的目的是结合半结构化存储策略，将属性数据合并到其所关联的IFC实体对象的数据中，映射为JSON数据，并结合ElasticSearch的action命令，将JSON数据嵌入到action命令中，形成JSON文档，为存储做准备。属性数据处理流程图如图3-3所示。由于IFC标准的属性类型丰富、结构复杂、类型嵌套、与IFC实体关联机制复杂等特点，在封装为JSON数据时，需要进行特殊的处理。对于属性数据模型和半结构化存储策略，将在第4章详细介绍。

图3-3 属性数据处理流程

1. 空间关系数据处理

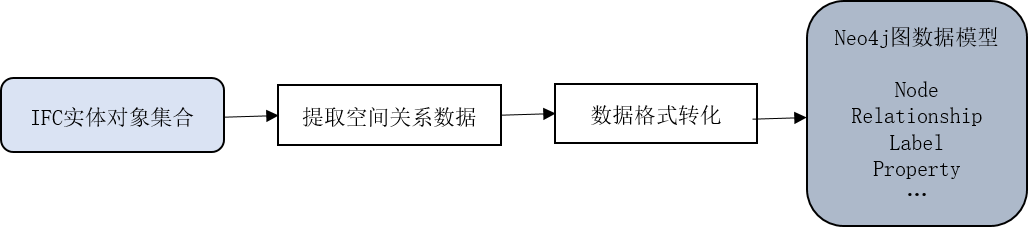
此模块的主要目的是提取IFC实体对象的空间关系数据，转换为Neo4j的节点（Node）、关系（Relationship）等数据类型。关于IFC空间关系数据模型将在第5章进行详细介绍。

图3-4 空间关系数据处理流程

### 存储模块

存储模块分为两部分：将属性数据索引到ElasticSearch集群中；将空间关系数据存储到Neo4j图数据库中。具体内容如下：

1. 基于ElasticSearch实现属性数据的半结构化存储。

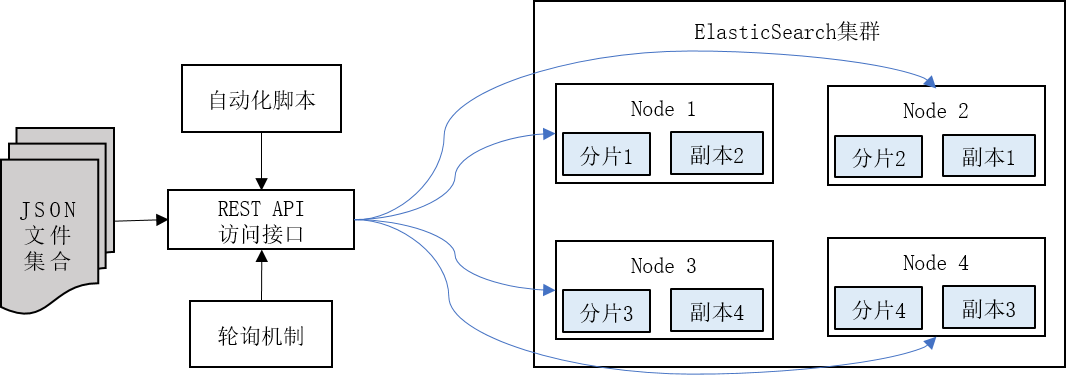
此模块的主要目的是实现批量索引JSON数据到ElasticSearch集群。ElasticSearch 提供了一套非常强大并且易操作的 REST API 用于集群的访问和索引的CRUD操作。本模块通过REST API，同时基于BULK批量导入机制，实现批量索引数据的目的。由于在数据处理模块，JSON数据被组织成以IFC类型名分类的JSON文件中，形成的JSON文件根据IFC文件中出现的不同的对象类型可能会有100多个到600多个。需要编写自动化脚本进行JSON文件的批量索引操作。整个批量索引请求都需要由接受到请求的节点加载到内存中，该节点解析元数据（\_index、\_type、\_id），然后分发给其它的节点的分片，进行操作。对于每个节点来讲，该请求越大，其它请求所能获得的内存就越少。为了减少批量索引时对单个节点造成的性能下降，在发送批量处理请求时采用轮询机制，将请求循环分发到集群中的各个节点，分散解析元数据的压力，也明显提高了索引的效率。

图3-5 批量索引数据到ElasticSearch流程

1. 基于Neo4j实现空间关系数据的存储。

此模块的主要目的是实现IFC空间关系的图数据到Neo4j的存储。本文采用Neo4j的嵌入式版本，自定义访问接口，可以实现图数据的批量存储。同时，引入动态建立索引的算法，其基本原理是结合用户查询热词以及索引空间限制，动态维护索引。动态建立索引算法将在第5章进行详细介绍。另外，本文在第5章针对一种的特定路径查询场景进行了介绍。在这种查询场景下，使用基于BloomFilter的查询算法，效率有明显地提升。

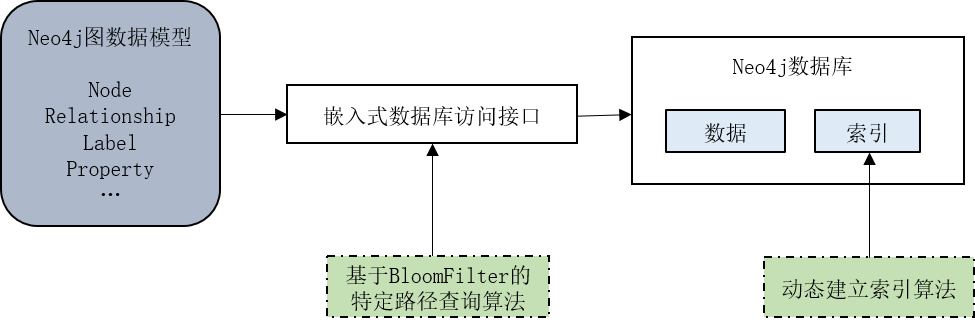


图3-6 Neo4j存储模块设计

# ElasticSearch存储关键技术研究

IFC标准通过定义属性类型、属性集类型为描述建筑构件提供了更加灵活的方式。属性类型数据可以表达丰富的属性信息，如墙的功能、结构、材质、吸收率、粗糙度等，在建筑初步设计、设计变更、施工图设计、施工实施等阶段都具有重要的应用意义。如果将属性信息独立存储，那么提取某个IFC实体对象的属性数据就需要进行多表连接，效率十分低下。

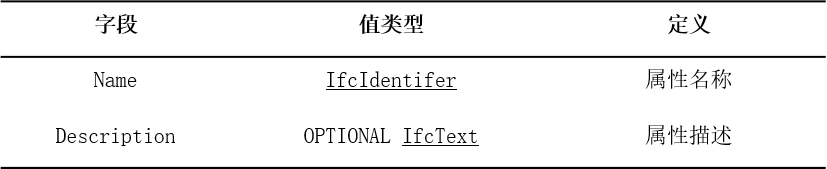
本文提出基于ElasticSearch的半结构化存储方式，将属性数据合并到IFC实体对象的数据中，避免了多表连接的效率问题。由于IFC实体的属性数据具有类型丰富、结构复杂、与IFC实体关联关系复杂的特点，将属性数据映射到JSON数据的过程需要针对不同的属性集定义进行分别处理。

本章将首先介绍IFC标准的属性数据模型，包括属性、属性集、和属性集与IFC实体关联机制的相关内容。然后介绍半结构化存储属性数据的算法，包括半结构化存储流程、属性数据到JSON数据映射的相关内容。

## IFC实体的属性数据模型概述

### 属性

属性是对事物以及概念描述的一个基本单位。在IFC标准中，除了IFC实体类型本身的各个字段可以描述IFC实体对象，还定义了不同的属性类型，为描述IFC实体对象提供了一种更加灵活的可扩展的方式。这些属性类型均继承自抽象基类IfcProperty。表4-1分别给出了抽象类IfcProperty的正向字段定义。

表4-1 IfcProperty定义

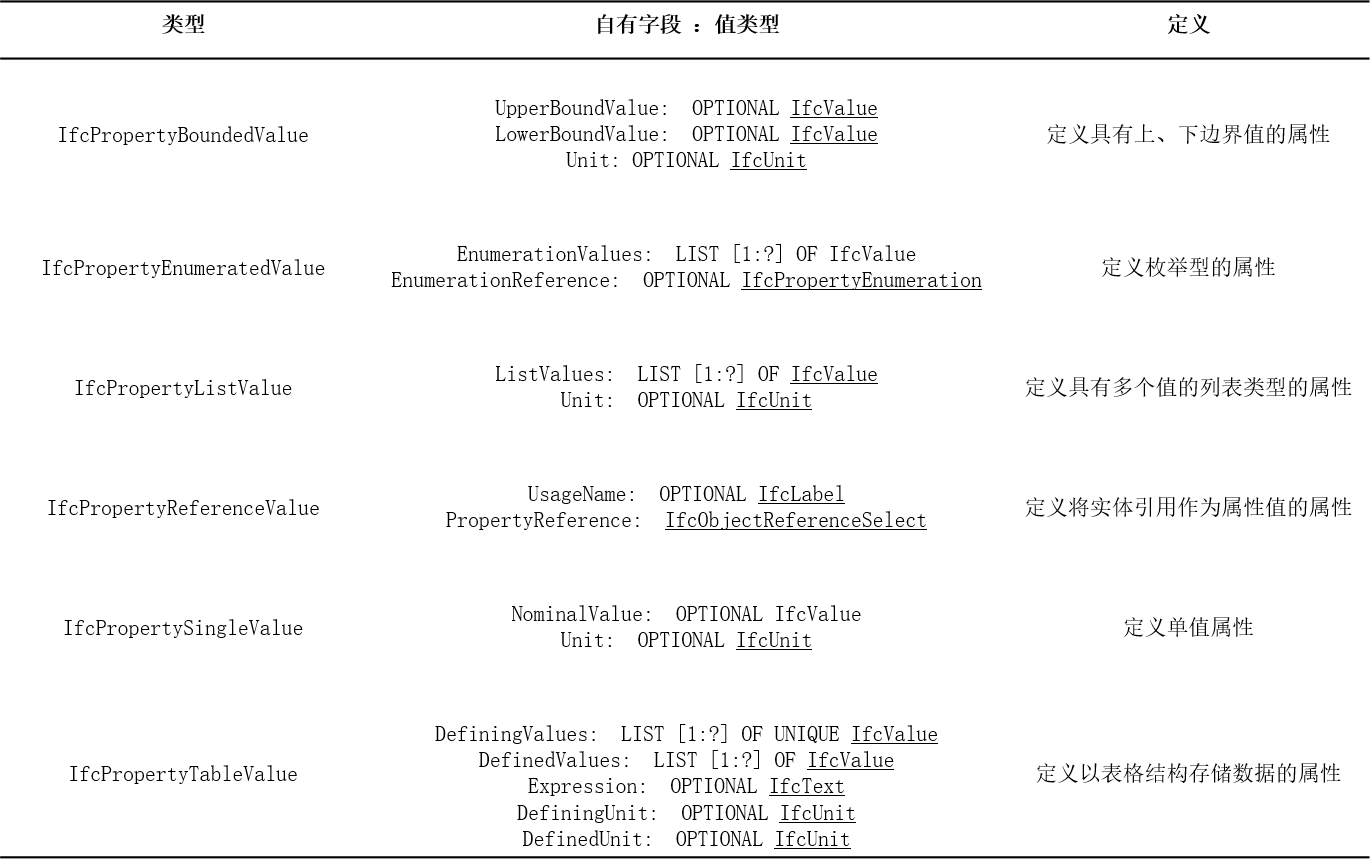
根据属性的值类型的不同，IfcProperty具有不同类型的派生类。图4-1给出了IfcProperty的派生子类图，图中的斜体内容表示此类是抽象类。

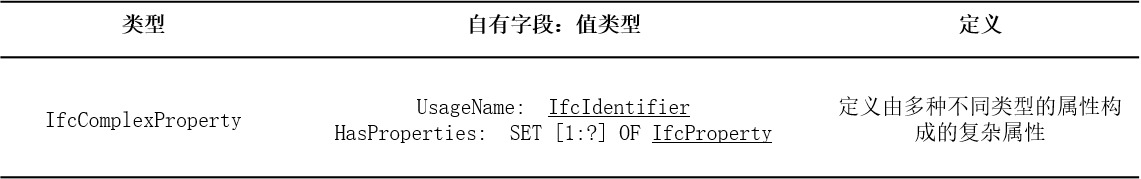


图4-1 Property派生子类图

IfcSimpleProperty定义简单属性类型，它的派生子类定对应定义了不同的属性值。它们的自有字段的定义如表4-2表示，可以看到，自有字段的值类型比较复杂，有定义类型（Defined Type）如IfcValue、IfcText等，有枚举类型（Enumeration）如IfcPropertyEnumeration，有选择类型（Select Type）如IfcObjectReferenceSelect，也有实体类型（Entity Type）如IfcUnit。

IfcComplexProperty定义复杂属性类型，它包含一个不同类型的属性的集合，也可以嵌套复杂属性类型。它的自有字段及定义如表4-3所示。

表4-2 IfcSimpleProperty派生子类的自有字段及定义

表4-3 IfcComplexProperty的自有字段及定义

### 属性集

IFC标准用属性集来组织对象的属性。IFC标准定义了不同的属性集类型。属性集的分类如图4-1所示。所有的属性集可以分为静态属性集和动态属性集，动态属性集又分为预定义属性集和自定义属性集。所有的属性集类型都继承自抽象基类IfcPropertySetDefinition。IfcPropertySetDefinition派生子类图如图4-2所示。

1. 静态属性集

以自有字段的方式定义一组属性，只能用来描述特定IFC实体类型。静态属性集的类型及定义如表4-4所示。其中，以IfcDoorLiningProperties为例，其自有字段定义如表4-5所示。

1. 动态属性集

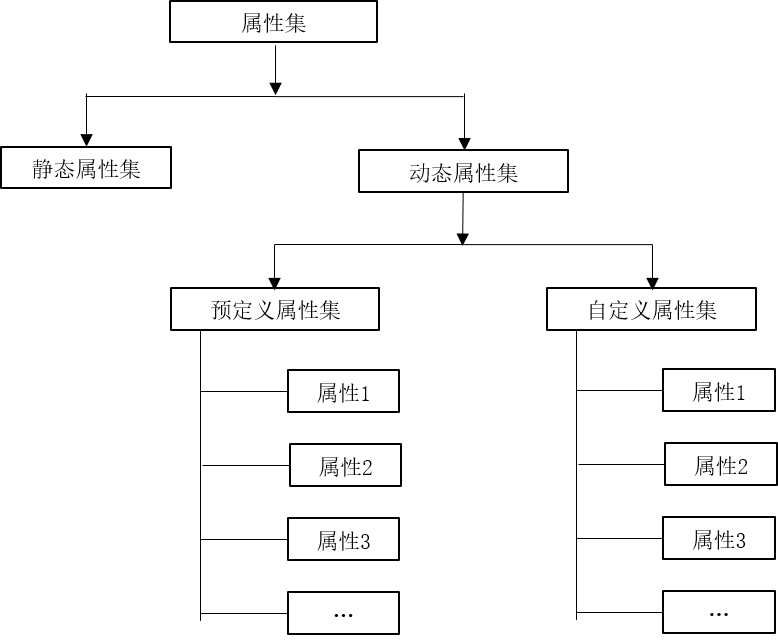
以包含一组不同的属性类型的方式来定义属性集。属性集类型是IfcPropertySet，其字段定义如表4-6所示。

* 1. 预定义属性集

由IFC标准定义，属性集名称以“Pset\_”开头，包含一组预定义的属性类型集合，用来描述特定的IFC实体。以Pset\_RailingCommon为例，其适用于实体类型为IfcRailing，属性定义如表4-7所示。

* 1. 自定义属性集

可以由用户自定义属性集名称和一组属性类型集合，它并不限于描述某一类IFC实体，具有更强大的表达能力。



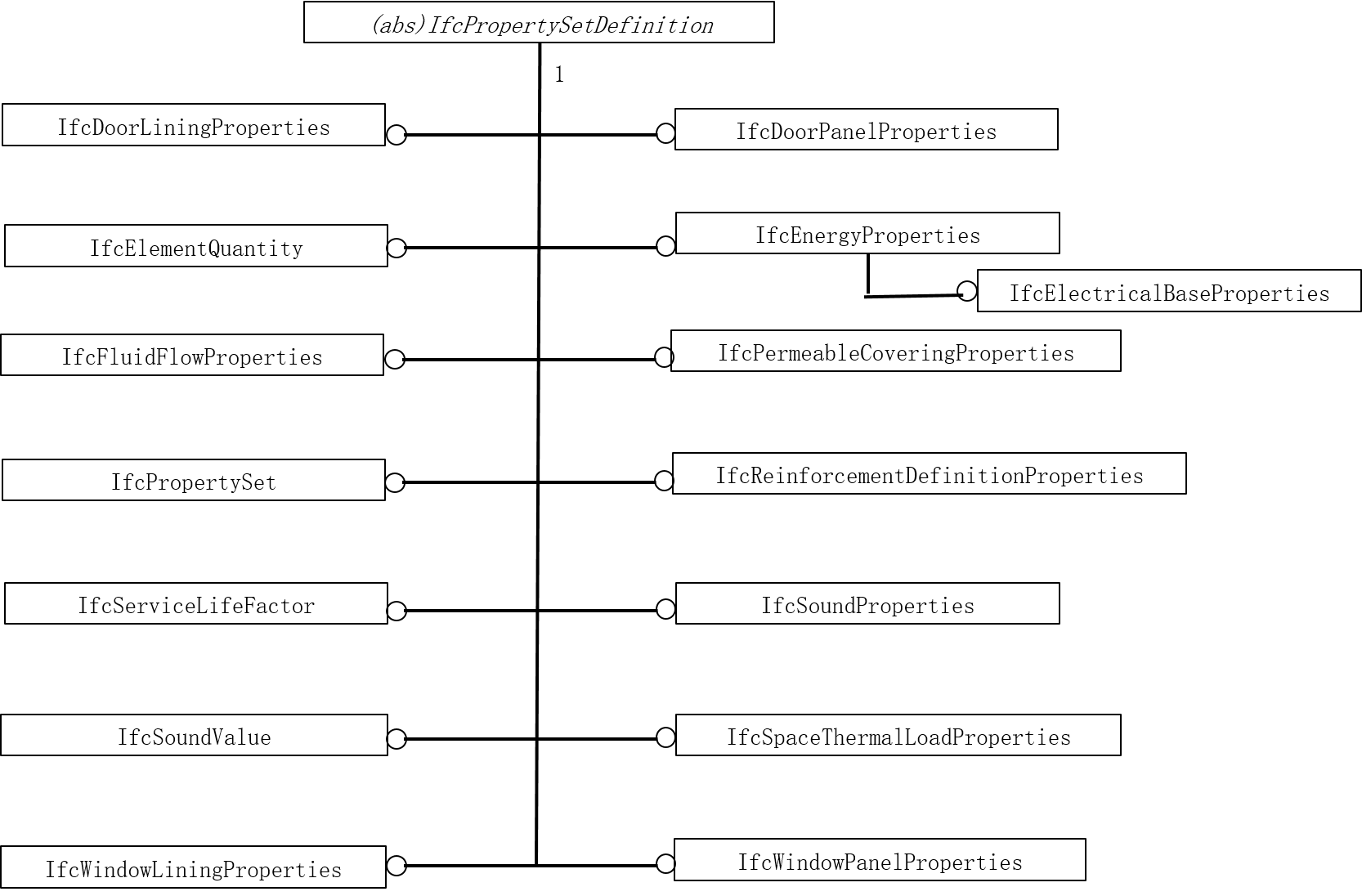
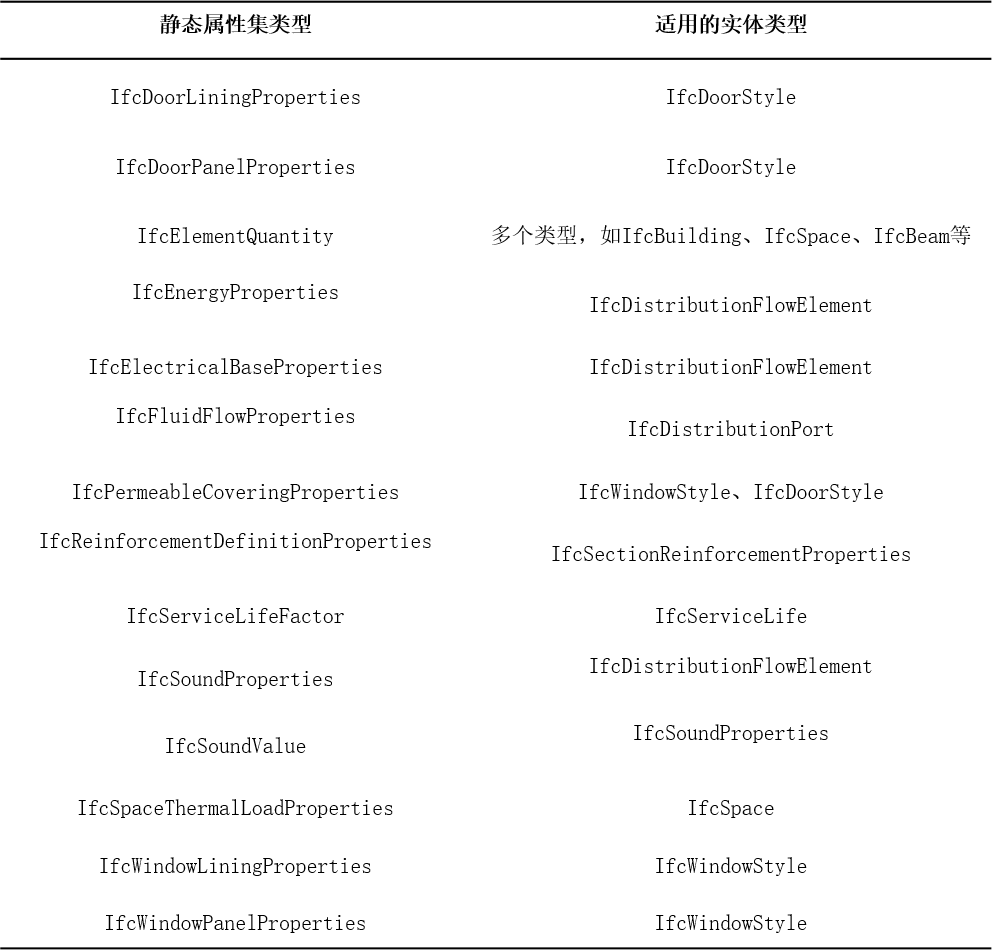
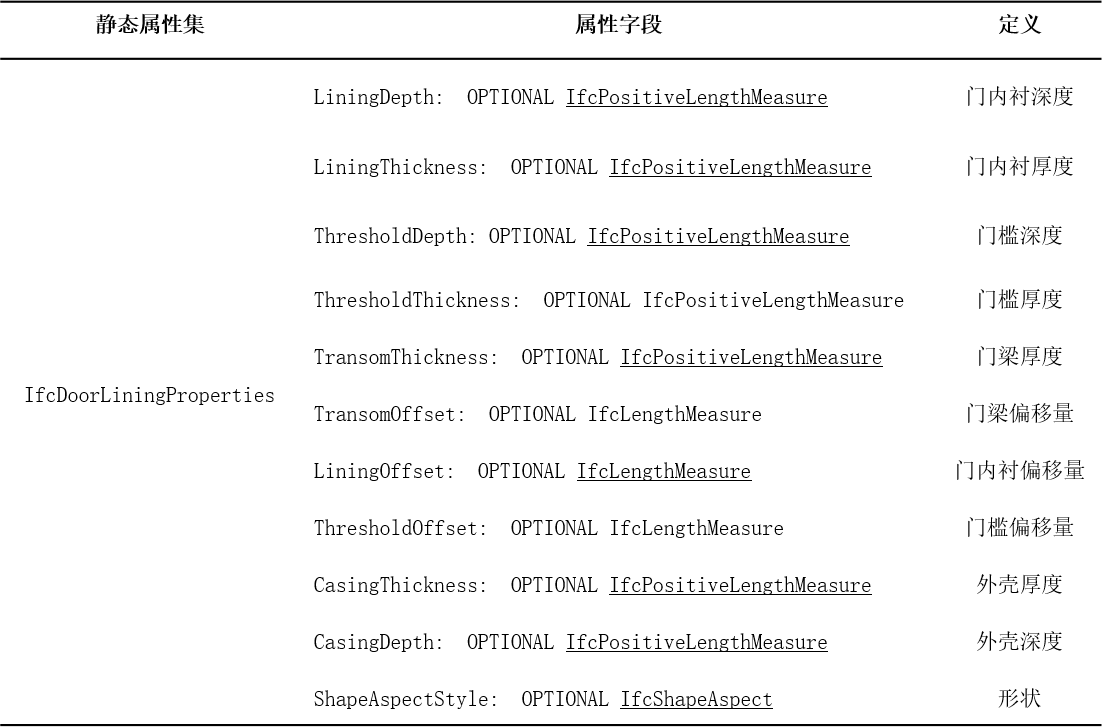
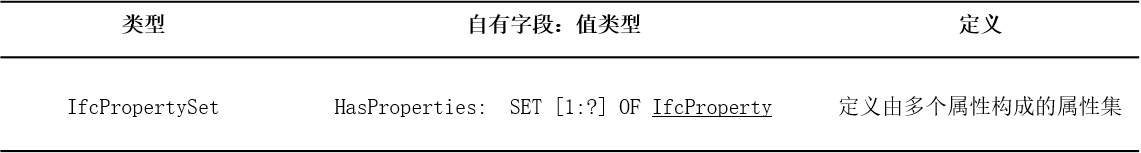
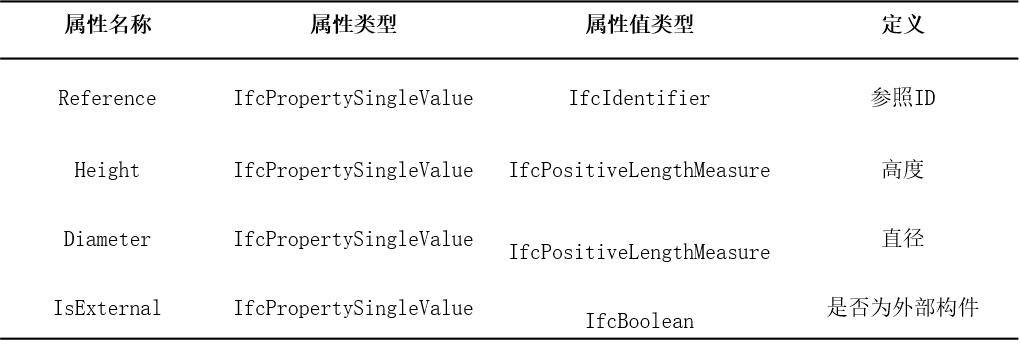
图4-1 属性集分类

图4-2 IfcPropertySetDefinition派生子类图

表4-4 静态属性集类型及适用的实体类型

表4-5 IfcDoorLiningProperties自有字段定义

表4-6 IfcPropertySet自有字段及定义

表4-7 预定义属性集Pset\_RailingCommon的属性定义

### 属性集关联机制

在IFC标准中属性集描述的信息通过以下两种方式关联到IFC实体：

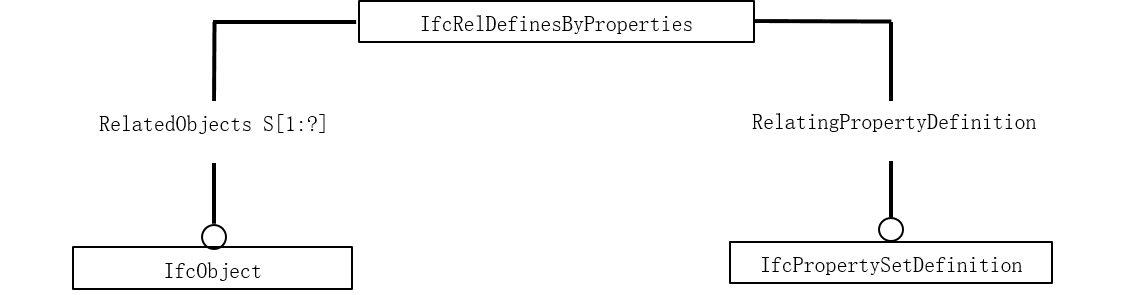
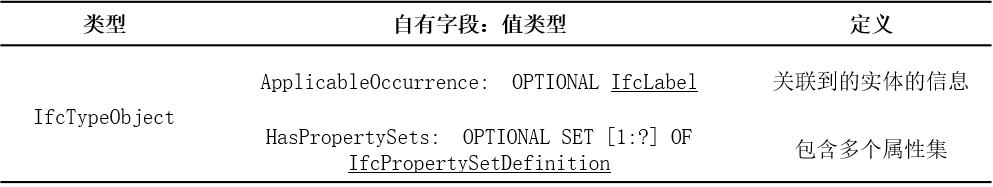
1. 通过IfcRelDefinesByProperties关系实体将IfcPropertySetDefinition对象与IfcObject对象关联。多个具有相同属性集的IFC对象可以通过同一个IfcRelDefinesByProperties关系实体关联到一个属性集，这样可以实现属性集的重复使用。同时一个IFC对象可以通过多个IfcRelDefinesByProperties来关联到不同的属性集，这样就可以借助多个属性集来表达丰富的属性信息。IfcRelDefinesByProperties关联机制如图4-3所示。

图4-3 IfcRelDefinesByProperties关联机制

1. IfcTypeObject的HasProperties字段直接引用一个或者多个IfcPropertySetDefinition对象，如表4-8所示。

表4-8 IfcTypeObject关联属性集

### 属性数据特点总结

IFC标准通过属性、属性集的定义提供了灵活的描述IFC实体的能力。总结属性数据的特点如下：

1. 类型丰富，结构复杂。属性包括简单属性和复杂属性，复杂属性支持嵌套。涉及到的属性集定义包括静态属性集、动态属性集，动态属性集又包括预定义属性集和自定义属性集。
2. 与IFC实体关联关系复杂。多个IFC实体可以关联到一个相同的属性集，一个IFC实体也可以与多个属性集关联。

以上特点为属性数据的存储带来了一定的难度。在4.2小节，本文将介绍如何进行属性数据的存储。

## 半结构化存储算法

半结构化存储的目的是将属性数据合并到IFC实体数据中，以避免提取属性数据时的join问题。半结构化存储的过程需要将复杂的属性信息映射为JSON数据格式。基于对IFC标准中的属性数据模型的研究可知，属性集与IFC实体的关联方式有两种，需要针对这两种方式，分别做不同的JSON映射处理，关键涉及到对于IfcRelDefinesByProperties对象和IfcTypeObject对象的处理。而对于其他IFC实体对象，则采用规范化的存储方式。

### 存储流程

半结构化存储算法的流程图如图4-4所示：

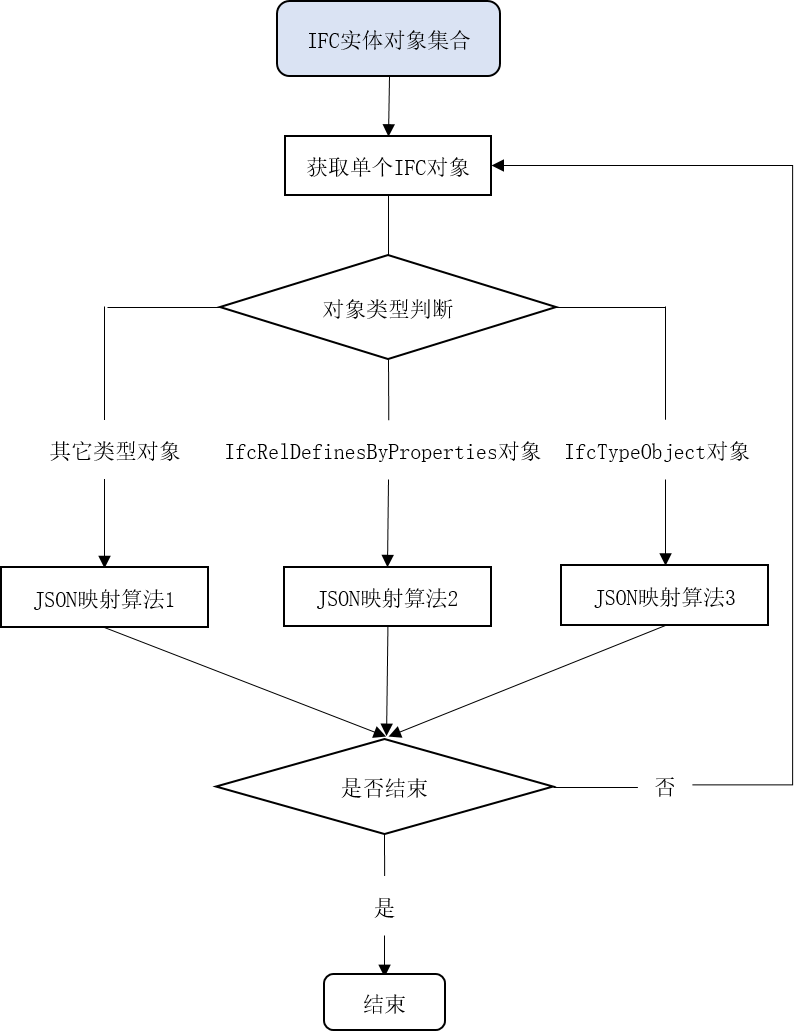


图4-4 半结构化存储算法的流程图

### JSON映射算法

JSON映射算法主要解决不同类型的属性数据如何合并到属性集数据中以及属性集数据如何合并到IFC实体数据中。

JSON映射算法的设计需要结合3点内容：JSON数据格式、ElasticSearch关于JSON存储的要求、属性数据存储需求。以下是详细介绍：

1. JSON (JavaScript Object Notation)是一种轻量级的数据交换格式。JSON采用完全独立于语言的文本格式，易于人阅读和编写。JSON数据格式有如下特点：
   1. 数据在键值对中；
   2. 数据由逗号分隔；
   3. 花括号保存对象；
   4. 方括号保存数组；
   5. 值的类型可能是数字、字符串、数组、对象等；
2. ElasticSearch关于JSON数据存储的要求：
   1. 同一索引不同类型下的同名字段的值类型要相同；

在ElasticSearch索引中，所有字段保持一个单一的、扁平的模式，所有的类型最终都共享相同的映射。一个特定字段可以映射成 string 类型也可以是 number 类型，但是不能两者兼具。

* 1. 索引中的字段不宜过多；

ElasticSearch会实时地将字段信息同步到集群中的每个节点。当字段过多时，如果此时索引新字段，将会严重影响集群的性能。ElasticSearch5.0及以上版本中，一个索引默认最多支持1000个字段。

1. IFC实体属性数据的存储需求：
   1. 数据不丢失；
   2. 字段名便于查询；

结合以上3点内容，将属性类型的数据映射为JSON数据的策略可以用如下的三个算法来描述：

**算法4-1**  JSON映射算法1

输入：非属性相关数据类型对象，如IfcOrganization、IfcPerson等

输出：JSON字符串对象

步骤：1) 增加元数据字段“\_index”、“\_id”，分别表示索引名称、对象Id。

2) 根据对象的类型定义获取字段名，根据对象实例数据获取字段值，在解析对象实例的数据时已经保证了字段值和字段名是一一对应的。根据字段名和字段值构建JSON字符串对象。

3)值类型：基础数据类型，对于引用数据类型。

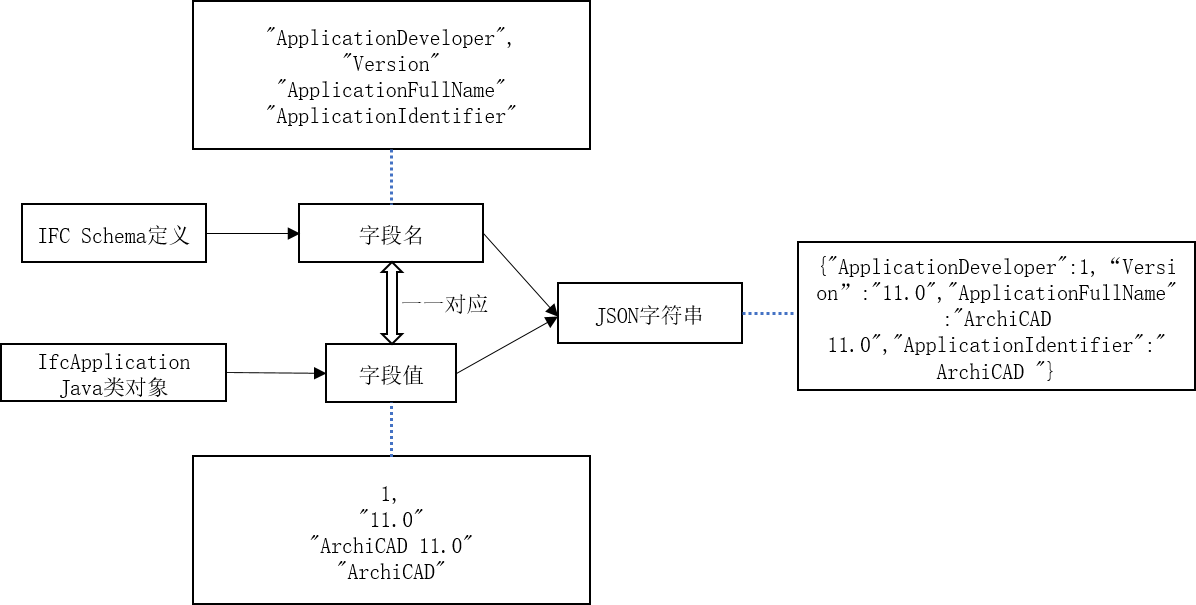
算法示例：以IfcApplication对象为例，其JSON生成过程如图4-5。

图4-5 IfcApplication对象的JSON生成过程

**算法4-2** JSON映射算法2

输入：IfcRelDefinesByProperties对象

输出：属性数据合并到IFC实体对象中的字符串

步骤：1) 获取IfcRelDefinesByProperties对象的“RelatedObjects”字段的值，得到关联的IFC实体对象集合E。

2) 获取IfcRelDefinesByProperties对象的“RelatingPropertyDefinition”字段的值，得到属性集IfcPropertySetDefinition对象。

3) 对于IFC实体对象e，e∈E，增加“Properties”字段，字段值类型为IfcRelDefinesByProperties对象数组类型。

5) 对于IfcRelDefinesByProperties对象中除了“RelatedObjects”字段以外的其它字段：

1. 增加“Id”字段、“Type”字段分别表示当前对象的Id值和类型名；
2. 如果是非引用字段，按照算法4-1生成JSON；
3. 如果是对于引用字段，则转到a)步骤对被引用对象做处理得到JSON，作为字段值。
4. 结束。

算法示例：图4-6展示了IfcWallStandardCase对象JSON数据的例子.



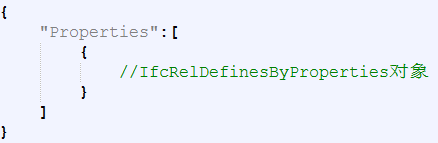
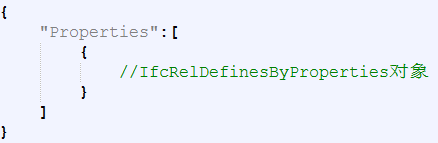
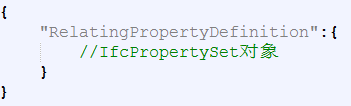
图4-6 IfcWallStandardCase的JSON数据示例

**算法4-3** JSON映射算法3

输入：IfcObject对象，多个IfcRelDefinesByProperties对象，多个IfcPropertySet对象，多个IfcPropertySingleValue对象，多个IfcPropertyBoudedValue对象，多个IfcComplexProperty对象。其关联关系如图4-12所描述：

输出：属性数据合并到IfcObject中的JSON对象

合并算法：

1. IfcObject的JSON对象增加“Properties”字段，值对应所有关联的IfcRelDefinesByProperties对象数组。形如：
2. IfcRelDefinesByProperties对象的RelatingPropertyDefinition字段，值为IfcPropertySet对象。因为一个IfcRelDefinesByProperties对象关联一个IfcPropertySet，所以此字段为单值字段。形如：
3. IfcPropertySet对象的HasProperties字段，值为IfcProperty派生类对象的数组。形如：
4. IfcProperty派生类中有复杂属性类型IfcComplexProperty，其HasProperties字段可以嵌套多个IfcProperty，形如：
5. 所有的对象增加Id字段表示行号，增加Type字段表示类型名，以保证信息不丢失。

以上处理中新增字段名有限，不会引起ElasticSearch在字段信息更新时的性能问题。同时，字段名固定，方便查询。

对于IfcTypeObject的HasProperties字段的合并算法如下：

1. 只需要加上它的子类就可以了。

可不可以给出算法流程图的。或者是给出例子。

# Neo4j存储关键技术研究

在绪论中本文对IFC标准进行了介绍。IFC标准的核心层中不仅定义了IFC对象的位置、几何形状等，同时也定义了IFC对象之间的关系。这些关系表现为空间结构上的组成、包含、连接等多种关系，与IFC对象在物理世界中的关系一致。根据空间关系信息可以方便地进行建筑子模型的查询。例如，查询某栋建筑的二楼的所有建筑构件，或者查询某个房间内的所有建筑构件。目前还没有针对IFC实体的空间关系数据存储的研究。

本文在深入研究了IFC空间关系数据模型的基础上，提出了基于图数据库Neo4j来实现IFC对象的空间关系数据的存储，从而可以实现根据空间关系数据提取子模型。另外，本文基于BloomFilter技术提出了一种针对特定空间关系的查询优化，提高了查询效率，同时对于空值查询能够快速失败，提前给用户响应。

本章的内容组织是：首先介绍IFC实体的空间关系数据模型，然后介绍基于Neo4j的存储技术，最后介绍基于BloomFilter的特定路径查询算法。

## IFC实体的空间关系数据模型概述

IFC标准划分为四个层次，包括资源层、核心层、共享层和领域层。其中核心层定义了IFC实体模型的整体框架，不仅包括IFC实体的位置和几何形状等，同时也定义了IFC实体之间的关系。这些关系表现为空间结构上的组成、包含等多种关系，与IFC实体在物理世界中的关系一致。例如一栋楼包含了许多楼层，而一层楼包含了许多建筑构件如墙、门、窗，某两面墙相互连接等。

在IFC标准中，IfcRoot是核心层及以上层次中全部实体类型的抽象基类。IfcRoot的派生子类有三种类型，分别是IfcObjectDefinition、IfcPropertyDefinition和IfcRelationship。而IFC实体之间的空间关系就定义在IfcRelationship当中。

IfcRelationship表示了不同的

IfcRelationship的派生类有IfcRelAssigns、IfcRelAssociates、IfcRelConnects、IfcRelDecomposes和IfcRelDefines五类。它们的定义如下：

* IfcRelAssigns及其派生类：表示Ifc实体（IfcObjectDefinition）与角色（IfcActor）、控制（IfcControl）、过程（IfcProcess）、资源（IfcResource）等之间的映射关系。
* IfcRelAssociates及其派生类：表示Ifc实体（IfcObjectDefinition）与约束（IfcConstraint）、文档（IfcDocument）、库（IfcLibrary）、材料（IfcMaterial）等的映射关系。
* IfcRelConnects及其派生类：表示Ifc实体对象（IfcObjectDefinition）之间基于某个标准的连接关系（connectivity）。例如，两面墙基于一条线相连接。
* IfcRelDecomposes及其派生类：表示Ifc实体对象（IfcObjectDefinition）之间的组成关系（composition）。例如，一栋建筑有五层楼。
* IfcRelDefines及其派生类：表示Ifc实体对象（IfcObjectDefinition）与属性类型（IfcProperty）、类型定义（Type）之间的映射关系。

根据这五类子类型的定义，只有IfcRelConnects和IfcRelDecomposes及其派生类涉及到Ifc实体的空间关系，因此在设计空间关系数据的存储模型时，重点考虑对这两类及其派生类的存储。

下面将举一个例子来介绍IFC实例对象之间的空间关系。

#94=IFCPROJECT('0DVPyye2z0deMtQa1pIata',#41,'\X2\987976EE7F1653F7\X0\',$,$,'\X2\987976EE540D79F0\X0\','\X2\987976EE72B66001\X0\',(#83,#91),#78);

#592=IFCSITE('0DVPyye2z0deMtQa1pIatc',#41,'Default',$,'',#591,$,$,.ELEMENT.,(39,54,57,601318),(116,25,58,795166),0.,$,$);

#679=IFCRELAGGREGATES('35aC7nYvTEcBYUkz$D5i0v',#41,$,$,#94,(#592));

#104=IFCBUILDING('0DVPyye2z0deMtQa1pIatb',#41,'',$,$,#32,$,'',.ELEMENT.,$,$,#100);

#683=IFCRELAGGREGATES('30UiOLvXbBv9bj$ovhCyLU',#41,$,$,#592,(#104));

#113=IFCBUILDINGSTOREY('0DVPyye2z0deMtQa2CjRCZ',#41,'\X2\68079AD8\X0\ 1',$,$,#111,$,'\X2\68079AD8\X0\ 1',.ELEMENT.,0.);

#687=IFCRELAGGREGATES('27PCKGLxT4mxtV9cw6mgBW',#41,$,$,#104,(#113));

#155=IFCWALLSTANDARDCASE('0o\_qQXI0r5UQtbU$t93HI7',#41,'\X2\57FA672C5899\X0\:CW 102-50-215p:309591',$,'\X2\57FA672C5899\X0\:CW 102-50-215p:412',#124,#151,'309591');

#672=IFCRELCONTAINEDINSPATIALSTRUCTURE('3Zu5Bv0LOHrPC10066FoQQ',#41,$,$,(#155,#367,#430,#550),#113);

#94代表项目，#592代表场地，#679代表项目（#94）“包含”场地（#592）。#104代表建筑，#683代表场地（#592）包含建筑（#104）。#113代表楼层，#687代表建筑（#104）包含楼层（#113）。（#155）代表墙，#367、#430、#550代表其它建筑构件，#672代表楼层（#113）包含这四个建筑构件。

这是一个典型的Ifc对象之间的空间包含关系的例子。如果用户需要指定特定楼层的某个建筑构件，这就需要通过Ifc对象之间的空间关系数据来进行查找。由于这种空间关系是使用关系对象来间接表示的，所以在一定程度上使得空间关系的查询并不容易。但是这种关系天然适合图数据库来进行存储。Neo4j是一个高性能的图数据库，它将整个复杂的关联数据集存储在一个大型的网络结构中，再采用一系列的图操作来实现对数据的管理和应用。

接下来本文将介绍基于Neo4j图数据库的IFC实体之间的空间关系数据的存储和查询相关技术。

## 动态建立索引算法

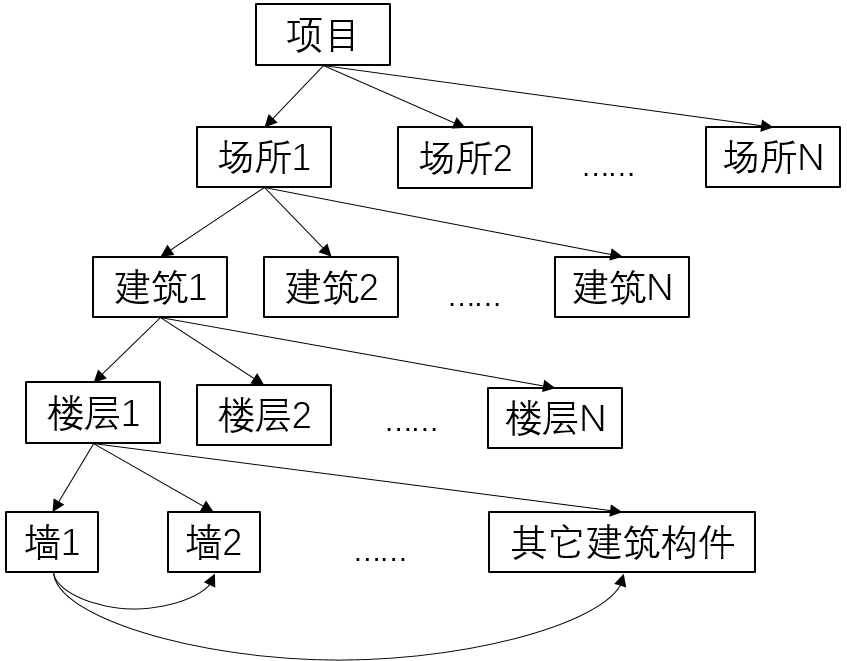
在现实世界中，建筑元素的关系可以是：某个项目包含很多场所，每个场所包含很多建筑，每个建筑包含很多楼层，每个楼层包含很多建筑构件，如门、窗、墙、梁、板、洞等等。而建筑构件之间也有多元化的关系，比如某两面墙相连接、

图5-1现实世界中的建筑元素的空间关系图

某面墙上有四扇窗等等。这些关系可以形象化地展示为图5-1.

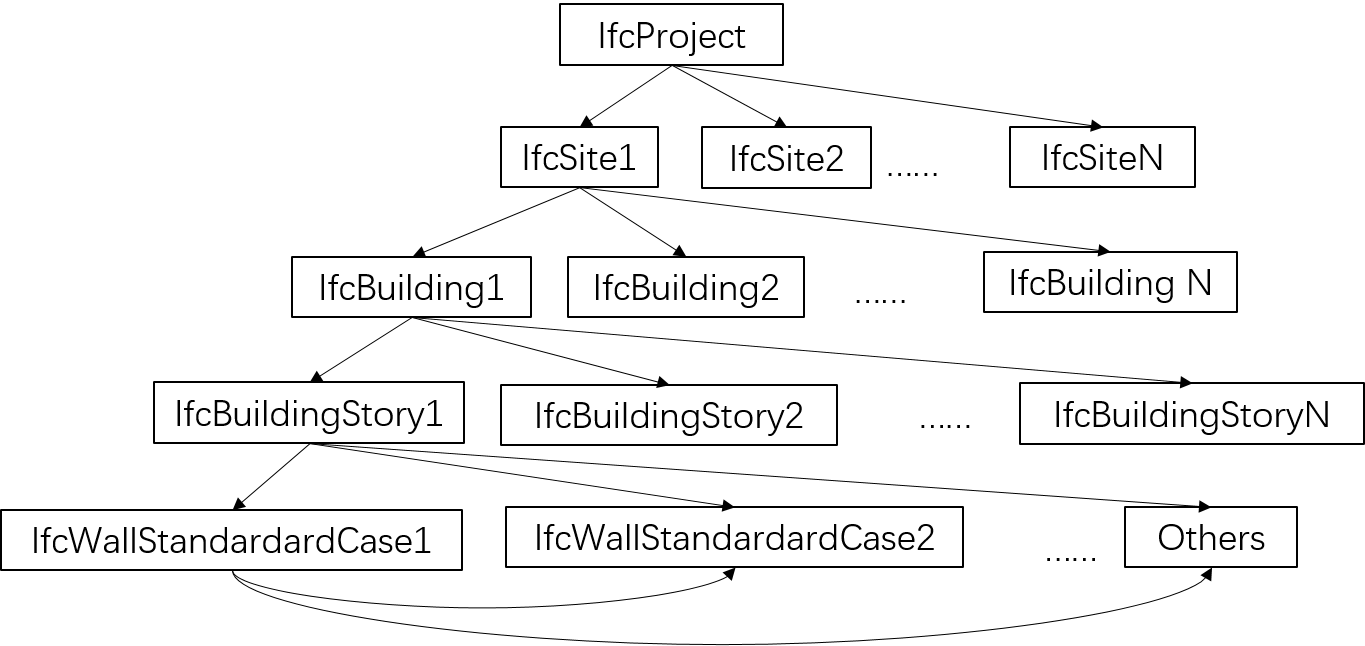
依照4.1小节对于IFC标准中关系的介绍，如果用IFC对象来表示上面图中的建筑元素，则对应的是图5-2。

图5-2 IFC对象的空间关系示意图

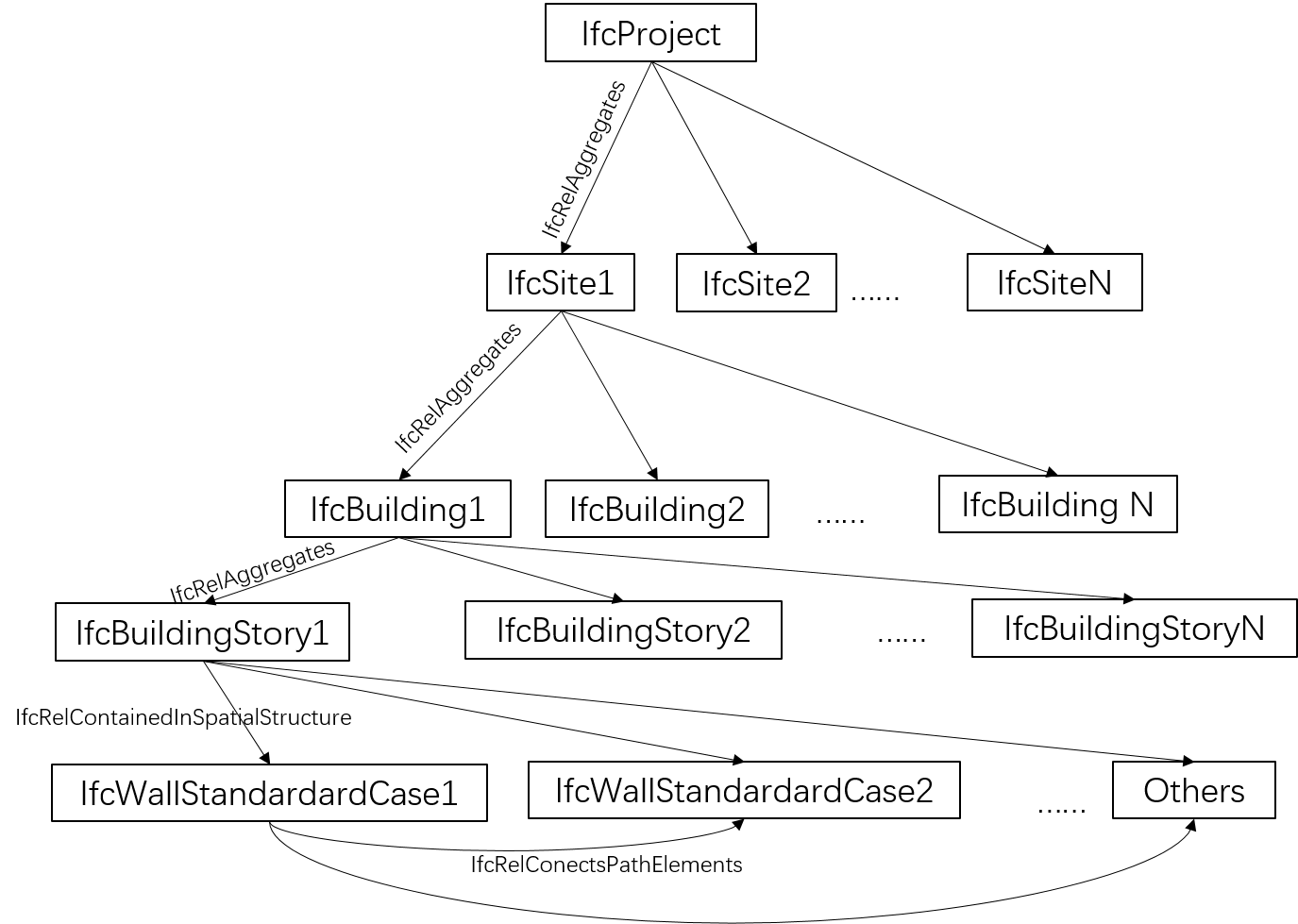
在上图中我们用一条带箭头的连线来代表IFC实体之间的空间关系。而这些关系在IFC标准中是用IfcRelationship的派生类来定义的。例如，IfcProject包含了很多IfcSite，这个包含关系用IfcRelAggregates来表示。而IfcWallStandardCase连接了另一个IfcWallStandardCase，这个连接关系用

图5-3 IFC实体的空间关系数据模型

IfcConnects关系类型来定义。

如果使用IfcRelationship派生类来替代上图中带箭头的连线，就是一个完整的IFC实体的空间关系模型，如图5-3。为了简洁，只标出了部分关系。

图5-3展示的模型可以直接对应到Neo4j的图数据模型。在Neo4j中，有两种基本数据类型，Node（节点）和Relationship（关系）。Node通过Relationship所定义的关系连接起来。同时可以在Node和Relationship上赋予key/value形式的属性，来表示详细的信息，以便对Node和Relationship进行查询。如果指定某个节点，从此节点出发，通过关系找到另外一个节点，这就形成了一条路径。这就是Neo4j的图数据模型。由此可知，图5-3中所展示的IFC实体的空间关系数据模型，天然适合Neo4j的图数据模型。每一个IFC对象对应图模型中一个Node，每一个IfcRelationship派生类对应图模型中的relationship。由此可以实现IFC实体的空间关系数据的存储，并依托Neo4j对于图数据的操作来进行IFC实体的空间关系的查询和应用。

## 基于BloomFilter的特定路径查询算法

前面几个小节已经讲述了如何基于Neo4j图数据库来进行IFC对象的空间关系数据的存储和查询。这已经能够满足用户对于某个对象或者某些对象以及它们之间的空间关系的查询。本小节将针对一种特定的路径查询场景提供优化方案。

### 特定路径查询场景及问题

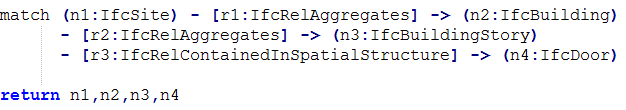
IFC标准中有一种特定的查询是，用户需要依次确定项目、场地、建筑、楼层，然后再找到楼层中满足一定条件的建筑构件，如门、窗、家具或其它零件。图3展示了这种关系在neo4j中的表示。在Neo4j图数据模型的定义中，从项目节点到建筑构件节点形成了一个路径，这个查询是一个特定的路径查询。其Cypher语句的简单形式如图5-7：

图5-7

又由IFC标准的定义可知，r1、r2、r3的类型是固定的，所以可以在Cypher表达式中省略掉类型。同时，用户需要对路径中的每一个节点加以属性值的约束，本文的示例暂且以“name”属性约束作为这个路径查询中对节点的约束条件。所以Cypher表达式应为图5-8：

图5-8

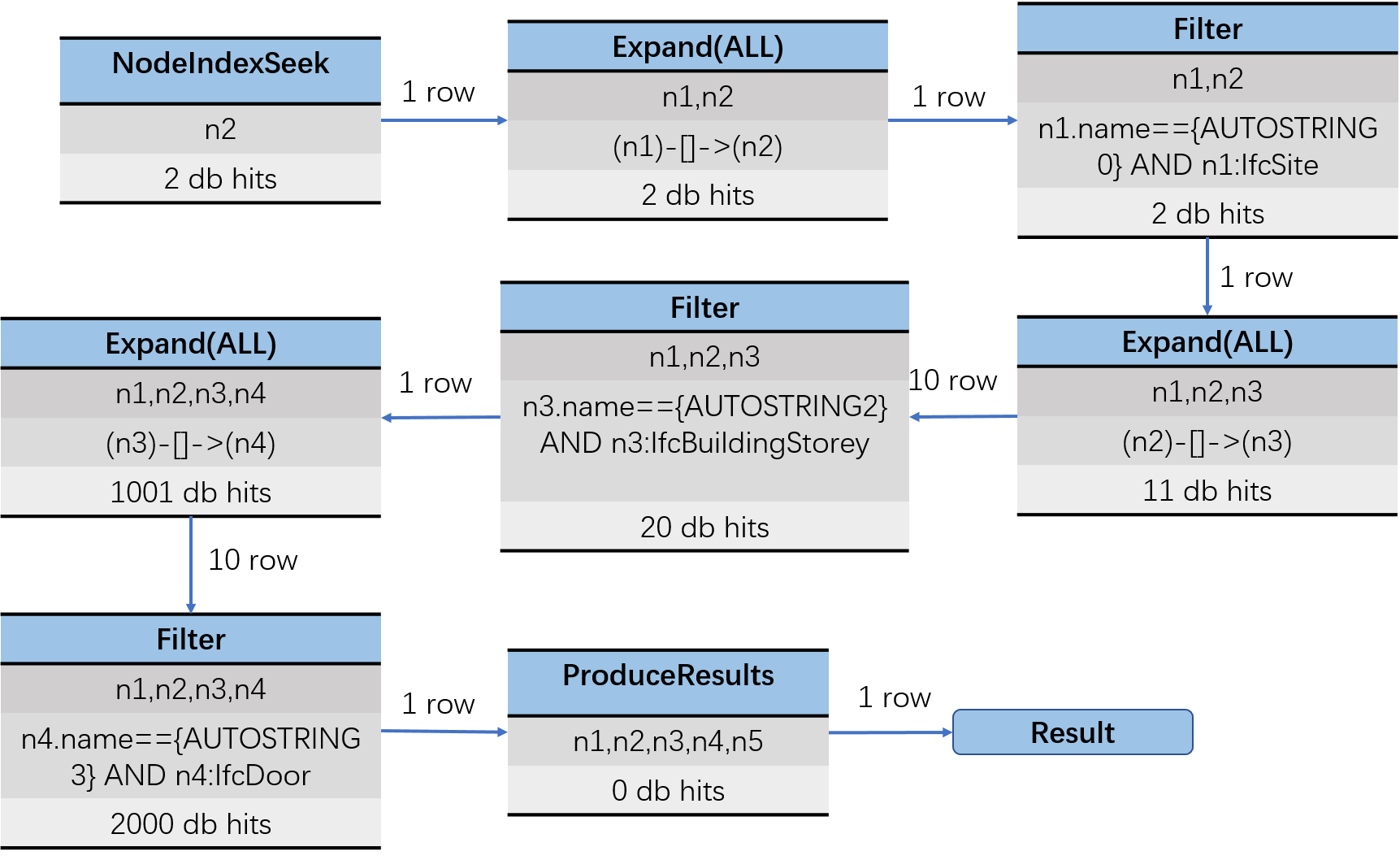
这个Cypher查询语句的完整的执行计划如图5-9：

图5-9

1. 执行NodeIndexSeek。访问索引“index on :IfcBuilding(name)”，返回节点n2。在这里假设只有一个IfcBuilding节点满足条件而返回。
2. 执行Expand(ALL)，访问指向节点n2的关系链表。因为一个IfcBuilding只有一个IfcSite指向它，也就是说指向n2的关系只有一个。所以访问db一次。
3. 执行Filter。过滤2）中关系所关联的节点的类型和属性，返回满足条件的IfcSite节点n1。
4. 然后从节点n2开始，依次执行Expand(ALL)和Filter过程。知道最后一个关系所关联的节点完成filter，找到满足条件的节点为止。
5. 返回结果。

分析这个过程，可以得出这样的结论：只有在整个执行计划的第一步使用了index，先找到一个切入点作为起始节点。而之后是通过访问关系链表数据找到关联的节点，而后对这些节点进行过滤。

这里有几个值得讨论的问题：

1. 没有充分利用index
2. 当节点所关联的关系数量增长时，链表结构的访问效率会成为影响查询效率的不容忽视的因素。
3. 由于是从起始节点依次开始访问关系链表和节点，很有可能出现当执行到最后一个filter环节时才发现没有满足条件的节点而返回空值，此时前面的查询操作已消耗了大量时间。

对此，本文提出用查询分解的方式，先将原始的查询分解为多个针对节点的查询，找出所有的满足条件的ID之后，然后将ID进行拼接，利用BloomFitler做结果过滤。

下面小节的安排是，先介绍关键的BloomFilter数据结构，然后介绍如何构建存储用于本小节所介绍的特定路径查询的BloomFilter，最后会介绍如何基于BloomFilter进行查询。

### BloomFilter原理

BloomFilter于1970年由Burton H. Bloom提出, 利用位数组标识一个集合，可以以较低的误判率来判断一个元素是否属于这个集合。因此这种数据结构适合应用在能容忍低错误率的场合。相较于传统的哈希函数映射和存储元素的方式，BloomFilter更加节省空间，从而能够满足数据量更大的应用场景。

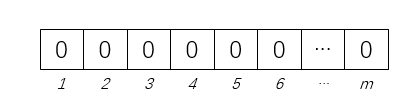
对于4.6.1小节中提到的特定的路径查询来说，如果我们能够把符合空间组成关系的IfcSite、IfcBuilding、IfcBuildingStory、IfcDoor（以及其它建筑构件）的ID按照顺序拼接起来，维护在BloomFilter当中。当查询来到时，将满足条件的各个节点的ID拼接起来，查看是否在BloomFilter当中，就可以知道，这个结果集是否存在。BloomFilter存在一定的误判。也就是说，如果这条路径并不在neo4j存储的图中，但是也有可能会判定为存在，此时用户得到的结果将会是由原始的Cypher查询得到的正确结果。同时，BloomFilter不会漏掉判断。这对我们是相当有利的，因为不会漏判意味着，只要判定这个字符串在BloomFilter当中，那么它一定存在。此时，我们只需要返回这个字符串就可以了。另外，BloomFilter用很少的内存就可以支撑上亿条记录，基本可以满足我们的存储和查询需求。

BloomFilter的核心思想就是利用多个不同的Hash函数来解决单一Hash带来的“冲突”。传统的Hash表利用同一个Hash，计算不同的字符串得到的值可能相同时，这就是“冲突”。如果要减少冲突，必须要将位数组的长度扩大。这样会消耗大量内存。为了解决这个问题，BloomFilter支持选择k个不同的哈希函数，这样就可以大大地减少冲突，同时保证低的内存消耗。

BloomFilter的原理要点：位数组、k个独立的Hash函数。

1. 位数组：

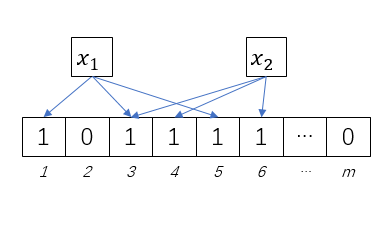
初始状态时，BloomFilter是一个包含m位的bit数组。每个bit都为0，如下：

1. 基于K个独立Hash函数，添加元素：

待添加的元素设为 。当加入任何一个元素x的时候，BloomFilter使用k个互相独立的Hash函数，得到k个Hash值，对应的范围内的k个位置。将这个m bit数组的对应k个位置的值置为1。如果这个位置被多次置为1，那么只有第一次会起作用。

例如，k=3，,,,

,,.

加入，则bit数组为：

1. 判断元素是否存在：

在判断元素y是否属于这个集合时，只需要对y使用k个Hash函数得到k个hash值，对应bit数组中k个位置。如果这k个位置都是1，那么就认为y是集合中的元素，否则不是。显然这不是100%正确的。如果y事实上并不在集合里，但映射到的k个位置刚好恰好已经置为1了，那么就会出现False Positive，也就是误判。

1. BloomFilter的参数：

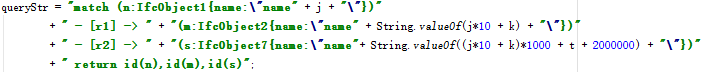
误判率与插入BloomFilter的元素个数n、bit数组大小m以及Hash函数个数k有关。研究表明，当Hash函数个数k时错误率最小。在错误率不大于E的情况下，m至少要等于才能表示任意n个元素的集合。

### 基于BloomFilter的查询

回顾4.6.1节中表达式2的Cypher查询所存在的问题，本文改进的想法和目标是：

1. 将查询分解为多个对节点的查询，从而可以充分利用index；
2. 由于1)步骤改变了查询计划执行的顺序，从而使得当节点本身不满足属性要求时可以快速失败，返回结果；
3. 使用BloomFilter的过滤机制，而不是链表的顺序访问机制，可以有效提升查询速度。

具体的算法逻辑是：

1. 将上面的Cypher查询分解为针对节点的Cypher查询。原始的查询格式比较固定，会是如下这种情况：

而分解后的查询会是如下这种情况：







1. 分别执行以上单个针对节点的查询，也就是3个NodeIndexSeek。如果结果集为空，则可以直接结束整个查询，返回给用户空值；
2. 如果结果集不为空，则将节点的ID字段拼接为一个字符串。
3. 已创建的BloomFilter包含了所有的可能的结果字符串。查看3）中的字符串是否包含在BloomFilter中。



如果在，则返回ID对应的节点。如果不在，则说明查询结果为空。

以上便是基于BloomFilter进行特定路径查询的算法及相关实现。

### 构建BloomFilter

根据4.6.1小节对于改进算法的描述可知，如果要想实现这样的查询，必须先基于IfcSite、IfcBuilding、IfcBuildingStory、IfcDoor（等）的空间组成关系，将所有的这些ID拼接字符串存入到BloomFilter当中。这需要在内存中解析IFC对象时就把它们的关系对应起来。为了达到这个目的，需要借助于以下几个Map结构：

1. Map<Long, Long> bdgToSite;

存储IfcBuilding的ID与其所在的IfcSite的ID的映射。

1. Map<Long, Long> storyToBdg;

存储IfcBuildingStory的ID与其所在的IfcBuilding的ID的映射。

1. Map<Long, Long> elmtToStory

存储IFC实体的ID与其所在的IfcBuildingStory的ID的映射。

基于这3个数据结构，可以构建一个IFC对象getStoryID()、getBdgID()、getSiteID()，得出IFC对象所对应的IfcBuildingStory、IfcBuilding、IfcSite的ID。然后按照从Ifcproject到IFC实体的空间组成顺序拼接ID为长字符串，插入到BloomFilter中。

关于上面的步骤，会有人有疑问：通过几个get方法不是已经可以找到上文所介绍的路径查询的结果了吗？的确是这样。但是问题就在于，Map结构非常占用内存。这也是本文为什么需要BloomFilter数据结构的原因，它能大大地缩小内存占用，从而可以常驻内存。

# 系统实验

## 属性数据相关测试

本文提出了基于ElasticSearch的IFC属性数据存储和查询方案。接下来将针对该方案进行测试，分以下内容进行介绍：

### 实验环境

本文搭建了具有4个节点的ElasticSearch集群，另有一台主机来访问集群。集群的服务器配置如下图：

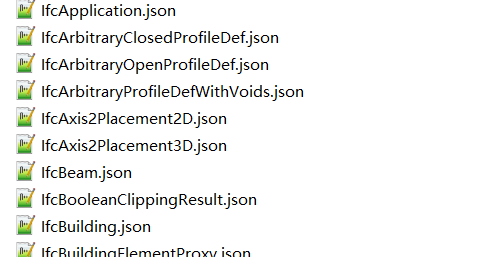
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IP | CPU | MEM | HDD | OS |
| 10.141.211.150 | Intel(R)Xeon(R) E5649 2.53GHz | 10G | 194G | CentOS 6.7 |
| 10.141.211.151 | Intel(R)Xeon(R) E5649 2.53GHz | 14G | 220G | CentOS 6.7 |
| 10.141.211.152 | Intel(R)Xeon(R) E5649 2.53GHz | 13G | 198G | CentOS 6.7 |
| 10.141.211.153 | Intel(R)Xeon(R) E5649 2.53GHz | 12G | 195G | CentOS 6.7 |

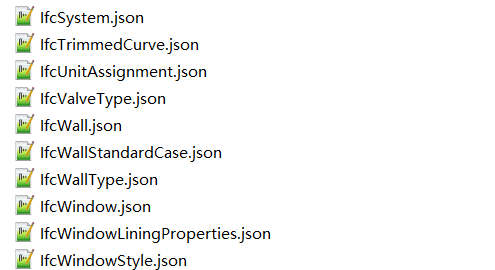
本次试验部署的ElasticSearch的版本为5.4.2,是近期更新的版本。

### 存储算法实验

受限于IFC文件来源并不多，本文将基于现有IFC文件生成测试数据。现有IFC文件大小为94M，是重庆研究院大楼的设计文件。其用IFC文件查看工具IFC Viewer呈现出来的效果如下：

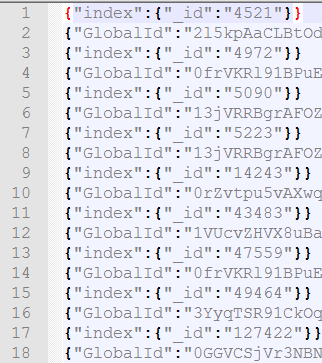
IFC文件解析后生成JSON文件160个，JSON数据总量为126M，部分文件截图如下：



**······**

下图是解析后的IfcWallStandard.json的示例：

下图是解析后的IfcWallType.json的示例：

 本文在解析IFC文件的每一行数据时为IFC对象分配新的ID，代替原来的行号。IFC对象之间存在互相引用，引用字段的值是被引用对象的ID。为了保持这种引用，需要引入map结构，来维护旧ID与新ID的映射。

新分配的ID对应ElasticSearch的\_id，从而索引为不同的文档。每解析一次，产生一倍的数据。试验共生成了约220万行文档数据，5个索引分片分别配置了一个副本，数据分布在集群中的4台服务器上。

### 系统可用性测试

## 空间关系数据相关测试

本文采用Neo4j来存储IFC对象的空间关系数据。下面来介绍基于Neo4j的索引和查询效率。

### 实验环境

本文基于嵌入式Neo4j 3.0.6版本的开发包来进行空间关系数据存储的实验。实验环境如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 运行环境 | CPU | MEM | OS |
| 单机 | Intel(R)CORE i5 | 8G | Win10 |

### 存储实验

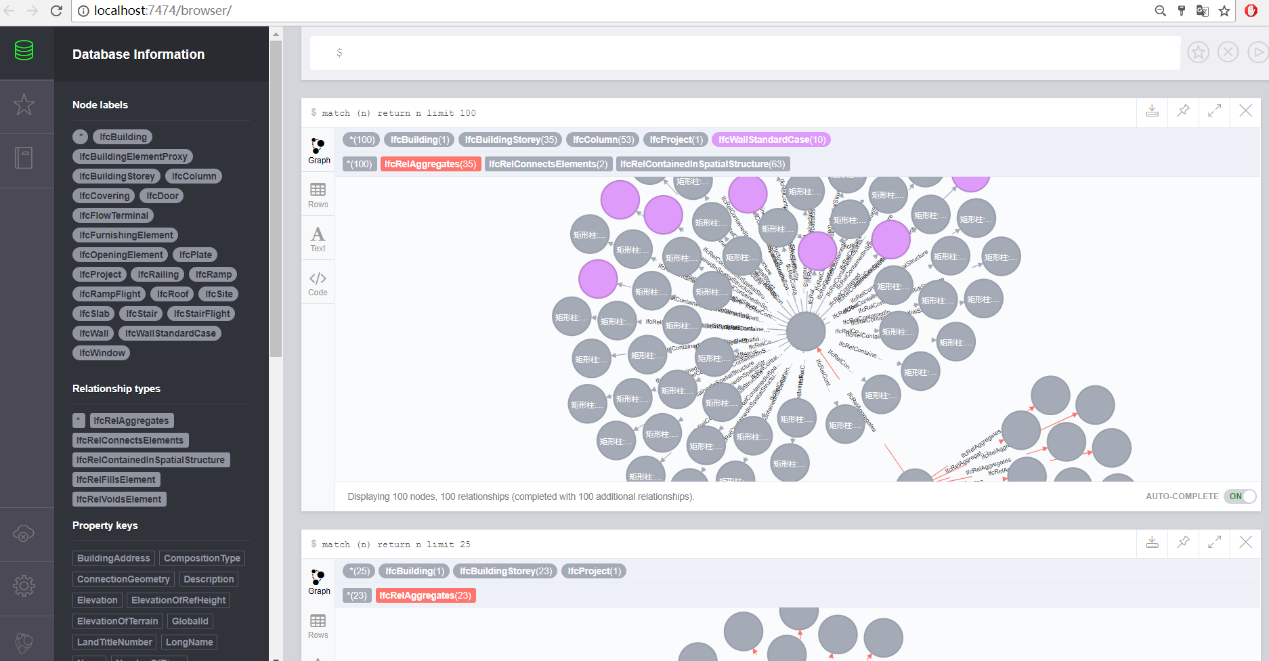
本实验使用BatchInsert方式批量索引节点和关系，经试验，插入的效率是平均每秒9708个Node，接近1万，是相当高效的。索引数据后，使用Neo4j社区版本的客户端连接数据库，并可以通过7474端口从浏览器访问数据库。其界面如图6-5：

图6-5

其中，左侧显示了节点的标签、关系的类型、属性等信息，右边可以使用Cypher语句进行查询。

### 基于BloomFilter的特定路径查询实验

在此试验中本文使用的数据集是参照了IFC空间关系而生成的约100万节点的测试数据。由于是特定路径查询，所以它的查询具有较为固定的格式。本文抽取了其中的10000条数据进行普通的Cypher查询。

同时，基于BloomFilter进行相同的查询，即分解查询、拼接ID、BloomFilter过滤③的方式，进行这10000次查询。对比时间消耗：

|  |  |  |  |
| --- | --- | --- | --- |
| 查询方式 | 第1次（ms） | 第2次（ms） | 第3次（ms） |
| Cypher（原生） | 33986 | 24051 | 24991 |
| BloomFilter | 11757 | 11972 | 11058 |

由以上三次试验，试验中原生Cypher最快时间24051ms较基于BloomFilter最慢时间11972ms还要多耗费10000ms。可知基于BloomFilter的查询速度明显快于原生Cypher查询方式。

### 系统可用性测试

# 总结与展望

## 总结

当今世界，一场新的制造业竞争已然拉开序幕：美国力促高端制造业回归、德国倾力打造工业4.0。而作为制造业大国，中国则推出了"中国制造2025"行动计划。建筑行业作为制造业的重要组成部分，得到了广泛关注，作为建筑行业的行业数据的标准，BIM行业得到了广泛关注。BIM系统是一种全新的信息化管理系统，目前正越来越多应用于建筑行业中。然而，随着时代的发展，智慧城市概念的提出，BIM数据有了长足的增长，以往的存储和查询手段已经渐渐越来越不能应付数据的长足提升。所以为海量BIM数据提供一种高效快速的存储和查询手段是有必要和有意义的。

结合国内外BIM数据库研究的发展方向，可以看到，BIM数据库研究的目标是支持大数据量的存储和查询、支持更为灵活的查询方式。

基于此目标，本文选择了开源弹性搜索框架ElasticSearch并展开了研究，通过增加冗余的方式，满足用户对于属性数据的高效查询，避免了多表连接的问题。在试验中，着重解决了IFC文件解析后的数据到JSON数据的映射问题，和ElasticSearch的索引优化等问题，最终实现了IFC实体的属性数据的高效存储和查询。

其次，本文还深入研究了IFC数据标准，提出了IFC对象空间关系数据模型的存储设计。本文选择了开源的Neo4j图数据库进行IFC实体空间关系数据模型的存储，从而在属性值查询的基础上，为用户提供了另外一种维度的查询方式，即空间关系（或者成为路径）的查询方式。

除此之外，本文进一步关注了一种特定的路径查询，并针对这种查询提出了基于BloomFilter的解决方案，使得查询效率得到了提升，尤其是在空值查询时能够提前失败，给用户响应。

## 展望

本系统基于弹性搜索框架ElasticSearch和图数据库Neo4j，实现了IFC实体属性数据和空间关系数据的存储和查询。但是目前，两个子系统在查询模块是独立的，用户只能通过各自的web界面来访问数据库，这带来了一定的不变。未来将考虑提供统一的访问界面。

另外，本系统中Neo4j的节点只存储了IFC实体类型本身的自有属性，这也是因为本文将研究的重心放在了IFC实体的空间关系信息的存储和查询上。如果用户需要满足一定空间关系的节点的某些热点属性（即用户关注的属性）的话，需要根据Neo4j结果集中的ID到ElasticSearch中查询数据。未来，可以考虑给IFC实体节点数据增加冗余的设计，将部分热点属性数据集成到节点数据当中去，这将使得用户可以直接在Neo4j找到热点属性数据，而不必在Neo4j做完关系查询之后再到ElasticSearch中查询。

# 参考文献

[1] Jorgensen K A,Skauge J,Christiansson P, Use of IFC Model Servers:Aalborg University and Aarhus School of Architecture, 2008

[2] 张洋，基于BIM的建筑工程信息集成与管理研究，清华大学，2009

[3] Sciences N I O B National Building Information Modeling Standard Verion1-Part1:Overview,Principles,and Methodologies,2011-9-10 ed 2007

[4] Fu C;Aouad G;Lee A IFC model viewer to support nD model application[外文期刊] 2006(2)

[5] Mell P;Grance T,The NIST Definition of Clouding Computing: National Institute of Standards and Technology

[6] Faraj I;Alshawi M;Aouad G.An Industry foundation classes Web-based collaborative construction computer environment:WISPER 2000(1)

[7] 陆宁，马智亮，利用面向对象数据库与关系数据库管理IFC数据的比较，清华大学学报（自然科学版），2012

[8] Kang H;Lee G,Development of an Object-Relational IFC Server 2009

[9] 刘强，张建平，胡振中，基于键-值缓存的IFC模型Web应用技术，清华大学学报（自然科学版），2016

[10] 余芳强，张建平，刘强，赵文忠，基于云计算的半结构化BIM数据库研究，土木建筑工程信息技术，2013(6)

[11] 张建平，BIM技术的研究与应用，2011

[12] 周雅莉，胡珉，喻钢，基于IFC标准的隧道工程信息传递研究，土木建筑工程信息技术，2015(12)

[13] 佐佐木达也，NoSql数据库入门，2012

[14] 岳莉，基于Lucene的全文检索系统的研究与应用，西安电子科技大学，2010

[15] 徐财应，基于Lucene的搜索引擎技术的研究与改进[D].中国科学技术大学，2014

[16] 陈亚杰,王锋,邓辉,刘应波, ElasticSearch分布式搜索引擎在天文大数据检索中的应用研究,天文学报，2016(3).

[17] Rafa l K, Marek R. ElasticSearch可扩展的开源弹性搜索解决方案. 时金桥, 柳厅文, 徐菲, 等译.电子工业出版社, 2015: 25

[18] 龙慧芬，移动社交网络中的数据库应用[J].山西青年管理干部学院学报，2013,26(3):106-108

[19] 王余蓝，图形数据库Neo4j的内嵌式应用研究[J].现代电子技术，2013,35(22):36-38

[20] 王余蓝，图形数据库Neo4j与关系数据库的比较研究[J].现代电子技术，2012,35(20):77-79.

[21] Huang H,Dong Z. Research on architecture and query performance based on distributed graph database Neo4j[C]. Consumer Electronics,Communications and Networks(CECNet),2013 3rd International Conference on .IEEE,2013,533-536

[22] Robin Hecht，Stefan Jablonski. NoSql evaluation：A use case oriented survey [A]. Proceedings of 2011 International Conference on Cloud and Service Computing [C]. 2011.336-341

[23] Sacco G M. Inverted index and inverted list process for storing and retriving information, U.S.Patent 8,738,631[P].2014-5-27.

[24] See R, Karshoej J,Davis D. An Integrated Process for Delivering IFC Based Data Exchage[J]. 2012-09-16