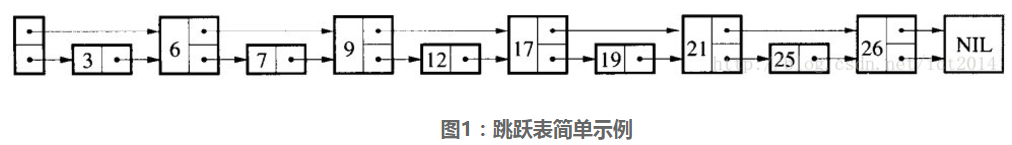
**基于Ne04j图数据库的社交网络数据的研究与应用 2016 湖南大学**

SkipList在leveldb以及lucence中都广为使用，是比较高效的数据结构。译文：跳跃表使用概率均衡技术而不是使用强制性均衡，因此，对于插入和删除结点比传统上的平衡树算法更为简洁高效。



重点分析了Ne04j图数据库目前所采用的多维索引，并且针对多维索引存在数据量很大时查询效率低下的问题，提出了将SKIP LIST跳跃表索引运用到Ne04j图数据库中的方法。

采用的新浪微博的数据。八爪鱼采集器。

我们从抓取的CSV格式的数据，进行了导入到Ne04j图数据库中，并建立相

应的关系，再导入到图数据库中我们利用SKIP LIST的插入算法将索引存入到链

表中。在这里我们是每七百条数据批量提交一次，大大提高了效率。

1.节点数、关系数、改进和为改进对应的各个指标的查询分析时间、存储空间的对比；

2.关系数据库执行路径深度查询的时间、neo4j执行路径深度查询的时间（假设100万人，每人有50个朋友）。

从论文当中的结果来看，千万级别的节点，估计索引文件在G级别。

e04j图数据的数据是以一种针对图网络进行过优化的格式保存在磁盘上【29】。



<GraphDatabases>(作者：IanRobinson)一书，描述了neo4j的存储模型！已下载！

Calculating the cost of graph database performance

The object cache is an on-heap cache that stores object in‐ stances of nodes, relationships and properties.

建议：增加缓存（操作系统缓存、jvm缓存）

<http://blog.csdn.net/huaishu/article/details/11713753> 有读取各个物理文件的结果展示！

1. Node[0,used=**true**,rel=9,prop=-1]
2. Node[1,used=**true**,rel=1,prop=0]
3. Property[0,used=**true**,prev=-1,next=-1,PropertyBlock[SHORT\_STRING,key=0,value=Neo122333],PropertyBlock[SHORT\_STRING,key=1,value=QQ],PropertyBlock[INT,key=2,value=100]]
4. Node[2,used=**true**,rel=2,prop=2]
5. Property[2,used=**true**,prev=-1,next=-1,PropertyBlock[SHORT\_STRING,key=0,value=Morpheus],PropertyBlock[SHORT\_STRING,key=1,value=QQ],PropertyBlock[INT,key=2,value=100]]

Neo4j的底层存储结构：

节点索引物理文件：neostore．nodestore．db, 长度固定9B，算是一个数组吧。查询成本O（1）。

neostore.nodestore.db.id件保持对未使用的记录的跟踪。

属性：neostore．propertystore．db，长度似乎也可变，默认41B。一个双向列表来存储的。是否可用+前一个属性Id+后一个属性Id+属性块32个字节。一个节点如果有多个属性，一个属性记录集无法存下，则通过下一个属性Id存储，最终通过上下属性Id完成列表连接。

neostore．propertystore．db．index属性索引文件，存储属性名称，对于每个属性的

值，记录包含一个指向动态存储记录的指针或内联值。

联系：store．relationshiDstore．db，长度固定33B。双向列表。

一般索引是在节点或者关系上的属性上加索引。

PropertyRecord ProperyBlock 属性块格式：属性类型（8B）+属性值（如果非基础类型占8B）

属性键与属性值分别存储在不同的文件中；

如果属性值大于默认长度，则需要动态存储，类似数据库BLOB字段的存储。存储较大无结构的二进制数据，存储在内部表空间。BFILE存储二进制文件，指向操作系统文件。

PropertyStore.encodeValue方法对属性数据进行编码处理。

通过生成的neo4j文件，输出节点、关系、属性了解他们之间的关系，数据存储结构的关系。

如何做到的？

Neo4j的多维索引

用Ne04j图数据库的后台控制的地址http：／／Iocalhost：7474／webadmin／#／index／，P33

本篇论文的数据集采用的是新浪微博的数据集，为了减少数据采集的时间，本文采用八爪鱼工具进行新浪微博数据的采集。采集的文件格式为CSV格式，这是为了方便使用JAVA代码进行读取内容。据集由两部分组成：社交关系数据，用户信息数据。

批量提交数据提升效率。

自动索引？？

Neo4j的索引是加载进内存了吗？应该是的，所以那本书的作者才说，需要加大cache的大小。

疑问：如何解决属性存储的问题？作者并没有给出相关实验。作者给出了skip list索引的查询算法，但是却没有给出如何利用这个索引进行数据统计，而是只给出了Neo4j的cypher统计方法。作者并没有说明索引存储空间大小是如何得到的。

我们的数据中关系并不比节点多，这是个问题？

还有我们的数据因为有父类子类的定义是不是也可以依照图数据库来进行存储研究？

是不是可以将需求分为两个主要的类别？体现为这两个需求场景：一个是对自身的属性的查询。一个是对空间关系的查询？

为什么不能够对一些资源实体也做一个存储呢？以往的解决方案中这些资源实体不是可以独立交换的单位，所以是直接存储在root实体的属性当中了吗。

**基于Ne04j的大数据组织检索研究与应用 2015 东南大学**

针对现有方法检索结果TOPK查准率较低这一问题，设计了一种面向Ne04j的全文检索算法。该算法首先提取Ne04i中节点的对应属性，经过中文分词后构建全文索引，然后通过引入向量空间模型及Lucene相关技术，计算查询关键词与待检索节点的相似度，最终实现检索结果按相似性大小排序。最后，利用图论相关知识挖掘数据间的关联关系，包括智能推荐、影响力分析、聚类分析及路径分析，最终完成了基于Ne04i的大数据组织检索系统，包括数据组织模块、数据检索模块以及数据分析模块。

1. 首先，引接海量规模的多源异构数据，将其转换为统一标准的JSON文件格式；其次，在基于Batch技术批量导入节点及节点间的关系的同时，建立标签及精确Exact类型索引，构建关系网络子图；最后，通过配置Ne04；集群及Mater-Slaver主从复合数据库，有效的组织存储大规模的数据。

2. 设计了一种面向Ne04j的全文检索算法。首先，以Neo4j节点对应属性作为研究对象，结合正向最大匹配MMSEG中文分词算法，切分节点属性文本，并以此构建基于Neo4j的全文索引库；其次，切分检索关键词，通过检索倒排表匹配查询节点；最后，引入向量空间模型，计算查询关键词与待检索节点问的相似度，并按照相似度大小倒序排列，小批量缓冲式返回相似性最大的TopK个节点。同时该部分结合查找字典算法，构建三叉Trie树并序列化存储，实现了智能提示及模糊搜索的功能，为进一步实现基于Neo4j的大数据组织检索系统奠定了良好的基础。

Lucene默认使用布尔模型确定哪些文档匹配上查询词，使用向量空间模型(VSM)来对这些文档进行评分。因此在基于Ne04j的大数据组织检索的研究中，通过改写Neo4j中的Lucene

接口，定制相关检索服务，可以实现检索结果的目标排序。

Neo4j索引配置文件config

其中，为了确保检索的准确性，切分节点属性的分词工具必须和切分检索关键字的分词工具相一致。

在Ne04i数据库中，索引可以分为精确匹配(Exact)和全文检索(Fulltext)两种类型，缺省索引类型就是精确匹配Exact。

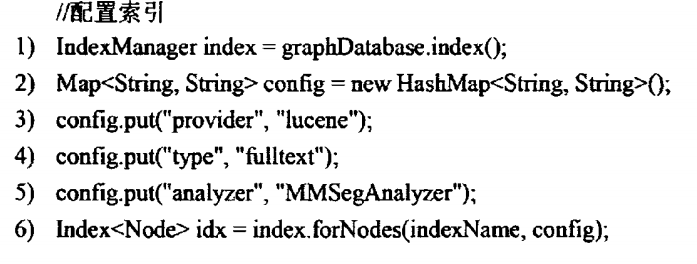
在创建Ne04i索引文件库前，必须配置索引信息config。在索引配置信息config中，

必须指定Ne04i中的索引类型以及中文分词算法。实际上，由于中文和西方语言结构上

的差别，将Lucene相关理论处理中文信息时，需要采用合适的方法对中文分词法。

MMSEG中文分词算法基于正向最大匹配算法（maximum matching，简称为MM算法）.

因此在本系统中采用MMSEG分词算法为Neo4j节点属性分词.



批量创建的索引的名称是什么？是什么配置呢？是否可以获取索引进行查看呢？

Exact类型和Fulltext类型的两种索引有什么不同？

索引读取器hldexReader、检索索引对象IndexSearcher和查询器QuervParser。

**基于Ne04j的电力大数据建模及分析 2016 华南理工大学**

还可以扩展出多种具有电力行业特色的拓扑算法，如基于节点连通性遍历的电网故障影响范围分析等。让我想起了对象关联时会影响到的对象？碰撞检测？

《一种基于BIM的建筑工程模型碰撞检测方法》

**基于Ne04j图数据库的时空数据存储 2015 重庆大学**

<https://my.oschina.net/costaxu/blog/108568>

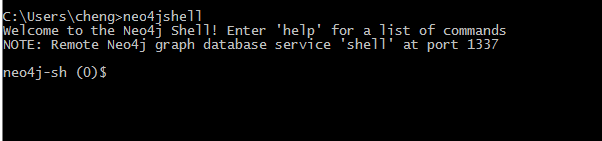
Mysql读写性能测试。作者的读是举例了最常用的给某个主键ID，查询某行数据。

<http://blog.csdn.net/holandstone/article/details/27211281>

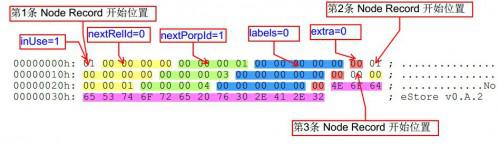
如同使用关系数据库一样，在使用 Neo4j 时，既可以使用 Neo4j 自身的 API，也可以使用第三方框架。Spring 框架中的 Spring Data 项目提供了对 Neo4j 的良好支持，可以在应用开发中来使用。Spring Data 项目把 Neo4j 数据库中的 CRUD 操作、使用索引和进行图的遍历等操作进行了封装，提供了更加抽象易用的 API，并通过使用注解来减少开发人员所要编写的代码量。

<http://blog.csdn.net/Sky_Money/article/details/8219349>

在配置Neo4j服务成功后，可以通过命令行打开Cypher查询界面，命令行中输入命令 Neo4jShell即可，如下所示：



<http://www.codeweblog.com/neo4j-%E5%BA%95%E5%B1%82%E5%AD%98%E5%82%A8%E7%BB%93%E6%9E%84%E5%88%86%E6%9E%90-8/>



Neo4j索引的物理结构：

<http://blog.csdn.net/u011697278/article/details/52462420>

以前，那时候neo4j还没有Cypher和server模式，只能使用java API操作Graph。

Manual Index是通过java API添加的。

**建manual Index的方法**：

IndexManager index = graphDb.index();

Index<Node> nodeIndex = index.forNodes( "nodes" );

Node node = graphDb.createNode();

nodeIndex.add( node, "name", "Thomas Anderson" );

缺点：建起来比较麻烦；滥用。优点：可以控制索引使用什么分词器。

在2.3.3版本里：In order to search for tokenized words, the query method has to be used. The get method will only match the full string value, not the tokens.

<http://www.cnblogs.com/nyzhai/p/4515102.html>

Neo4j创建自动索引。这里使用的是rest api的方式。是不是我也得了解一下？

[**利用Flex组件birdeye绘制拓扑关系图**](http://www.cnblogs.com/nyzhai/p/4465769.html)

<http://neo4j.com/docs/stable/batchinsert-examples.html>

关于BatchInserter。

动态的创建Label，relationshipType

Label personLabel = DynamicLabel.label( "Person" );

RelationshipType knows = DynamicRelationshipType.withName( "KNOWS" );

这里是这样添加index的：

inserter.createDeferredSchemaIndex( personLabel ).on( "name" ).create();

inserter.shutdown()一定要放在finally里面。

finally

{

if ( inserter != null )

{

inserter.shutdown();

}

}

还可以配置一些参数给Batch

Map<String, String> config = new HashMap<>();

config.put( "dbms.pagecache.memory", "512m" );

BatchInserter inserter = BatchInserters.inserter(

new File( "target/batchinserter-example-config" ).getAbsolutePath(), config );

从下面的这句来看，movies这个索引，可以直接调用query方法从索引中查询。

hits = movies.query( QueryContext.numericRange( "year-numeric", from, to ) );

<http://neo4j.com/docs/stable/indexing-batchinsert.html>

Index Batch Insertion，也可以由配置。跟普通的index配置一样。

Shutdown的时候都应该放在finally里面。

一些建议：不要flush too often；enable caching for keys你会经常用的。

cypher建schema Index：

CREATE INDEX ON :Person(name);

而上面说的似乎都是legacy index。通过在上面的文档目录里面抖索index关键词，

<http://neo4j.com/docs/stable/rest-api-indexes.html>

发现Legacy indexs is no longer the preferred way to handle indexes。

If you haven’t changed any indexing configuration, it means the indexes will be using a Lucene-based backend.

 schema based indexes, a feature that was introduced in Neo4j 2.0

*查看了一下，的确，schema index应该在schema文件夹下面，legacy index应该在index文件夹下面，所以这就说明，而现在是前者下面没有文件，只有label，而后者是有node、relationship的。Unindexed的index下面则没有东西。*

既然文档中都推荐用schema index了，那就用这个吧。

<http://blog.csdn.net/GeoSmart/article/details/51598710>

Neo4j索引笔记之SchemaIndex和LegacyIndex

 注意：使用legacy index查询往往需要一个start node？？？

legacy index采用非复合索引，更灵活，可以单独的访问某几个索引文件。schema index存储方式为复合索引（Compound Index），除了段信息文件，锁文件，以及删除的文件外，其他的一系列索引文件压缩一个后缀名为cfs的文件，即所有的索引文件会被存储成一个单例的Directory， 此方式有助于减少索引文件数量，减少同时打开的文件数量，从而获取更高的效率。比如说，查询频繁，而不经常更新的需求，就很适合这种索引格式。

文中的联合索引？？？？

创建自动索引？Neo4j 1.4引入了自动索引（automatic index），使用自动建索引，在config/neo4j.properties中配置。

<http://geosmart.github.io/2016/04/21/Neo4j%E4%B8%AD%E5%AE%9E%E7%8E%B0%E8%87%AA%E5%AE%9A%E4%B9%89%E4%B8%AD%E6%96%87%E5%85%A8%E6%96%87%E7%B4%A2%E5%BC%95/>

Neo4j中实现自定义中文全文索引

使用legacy index查询往往需要一个开始“节点”；并且它并没有能力提高查询的速度。

仅仅schema index有“标签”这个概念；legacy index完全没有“标签”的概念。

与legacy index不同之处在于，schema index能够提高查询速度。

schema index仅仅对节点是有效而legacy index允许“节点”和“关系”都被索引。“关系”索引的使用场景是很少的，并且通常都可以通过引入额外的节点解决问题。

Lucene：

一个索引可以包含多个段，段与段之间可以合并。一个段可以包含多篇文档。

Neo4j 已经内置了lucene document database。一个常见的应用场景是使用 lucene 搜索到一个节点，然后从其遍历或使用 cypher 查询。

Fulltext，如果想要检索匹配索引文档中的每一个word，那么你需要显式使用 fulltext 索引类型。

<https://stackoverflow.com/questions/9970193/how-to-store-tree-data-in-a-lucene-solr-elasticsearch-index-or-a-nosql-db?rq=1>

path enumeration or a Dewey Decimal

store a representation of the current node's location in the tree

把tree的结构存储起来，这种方法是，存储当前节点在tree中的位置。树是不移动和改变的。所以非常合适。