RDF 存储引擎的设计与实现

学　　院　计算机科学与技术

专　　业　计算机科学与技术

年　　级　2006 级

姓　　名　张 星

指导教师　张坤龙

2010 年 06 月 18 日

摘　要

随着近年来语义网的发展，互联网上出现了大量的 RDF（Resource Description Framework，资源描述框架）语义数据。为了使这些语义数据能够高效的发挥作用，需要研究高性能的 RDF 数据管理系统。RDF 数据管理系统的一个基本组件就是 RDF 存储引擎。

RDF 存储引擎的作用是解析符合 RDF 标准的 Turtle 格式的数据，将这些数据转换、压缩、编码，并为这些数据建立索引。RDF 存储引擎最终目标是建立一个全新的语义数据文件，这个数据文件支持标准 sparql 查询语句在大量数据的基础上高效执行。

关键词：RDF Store Engine;RDF 存储引擎

ABSTRACT

With the development of the Semantic Web in recent years, these is a large number of RDF (Resource Description Framework, RDF) semantics data. In order to make these semantic data work efficiently, we need to study the performance of RDF data management system. the RDF storage engine is basic component of the RDF data management system.

RDF storage engines resolves standard RDF data in Turtle format, and conversion, compression, encode the data. RDF storage engines ultimate goal is to create a RDF Store Engine ,this RDF Store Engine supports the standard sparql query to execute efficiently above on a large amount of data.

Key words: RDF Store Engine

目 录

第一章 绪 论 1
1.1 课题背景及其研究意义 1
1.2 国内外发展现状 1
1.3 课题研究的主要内容 1
第二章 需求分析 3
2.1 RDF 存储引擎的目标 3
2.2 传统 Turtle 格式的数据存在的问题 3
2.3 RDF 存储引擎的功能 4
第三章 RDF 存储引擎的设计 7
3.1 RDFStore 7
3.2 RDF 存储引擎的模块设计 8
第四章 RDF 存储引擎的实现 13
4.1 基础工具模块 13
4.2 RDF 数据文件解析模块 15
4.3 RDF 映射表建立模块 20
4.4 RDF 索引压缩算法 23
4.5 索引、数据文件的生成 26
4.6 小结 28
第五章 总结和展望 29
参考文献 30

外文资料

中文译文

致谢

第一章　绪论

1.1 课题背景及其研究意义

近年来，语义网技术引起了广泛的注意，并且在互联网上已经产生了较大数量的 RDF（Resource Description Framework,资源描述框架）格式的语义数据。为了使这些语义数据能够高效的发挥作用，需要研究高性能的 RDF 数据管理系统。RDF 数据管理系统的一个基本组件就是 RDF 存储引擎。

语义分析引擎效率的提升是个迫切而又艰巨的任务。计算机需要处理的数据是海量的，jena 等现有的语义分析引擎的效率还有待提高。然而，语义数据的数量还在不停的增长，数据的表示、压缩与组织是影响语义分析引擎效率的一个非常重要因素，在传统的数据表示方式不能满足高效查询要求的情况下新的语义数据存储引擎的研究非常重要。

本毕课题要解决的问题是如何以一种全新的方式将 turtle 格式的语义数据进行转换与组织，并使其可以支持高效的 sparql 语句查询与分析。这对于语义网高效性的研究有着重要的意义。

1.2 国内外发展现状

目前 IT 界对 RDF 的存储、索引和查询引擎的研究已经有几年了。惠普的 jena 框架目前已经比较流行，Oracle 公司还提供数据的 RDF 语义生命科学和企业集成支持。然而目前还没在大规模数据下非常高效的引擎出现，出众的 RDF Store 还需要进一步研究。脱离关系数据库的、不带有关系数据库的模式相对固定等缺点，用以支持高效语义搜索语句的 RDF 数据管理系统还处于很不成熟的阶段。

1.3 课题研究的主要内容

设计和实现的 RDF 存储引擎需要满足以下几个要求：

1、读入符合 RDF 标准的 Turtle 格式的数据，将其转换为三元组。这里需要设计和实现一个简单的词法分析器和语法分析器。

2、对字符串数据进行编号。编号时主语和宾语统一编号，谓语单独编号，编号过程中消除冗余。

3、建立一共 15 个索引。这 15 个索引包括 SPO、SOP、PSO、POS、OSP、OPS、SP、PS、SO、OS、PO、OP、S、P、O 等形式。由于索引的规模比较大，需要采用合适的压缩算法压缩索引。

4、建立一个映射表，映射表通过三元组中主语、谓语或者宾语的编号能够映射到原始的字符串上面。

5、建立一个数据文件，该数据文件由 RDF 数据转换而来，该文件数据库需

要采用特殊的数据组织形式以支持高效的查询。

6、使用 C#程序设计语言实现所设计的 RDF 存储引擎。

第二章　需求分析

2.1 RDF 存储引擎的目标

总的来说，RDF 存储引擎需要把符合 RDF 标准的 Turtle 格式语义数据以另一种方式存储起来，以这种方式存储的数据支持 sparql 查询语句的高效执行。图 2-1 展示了系统的需求。

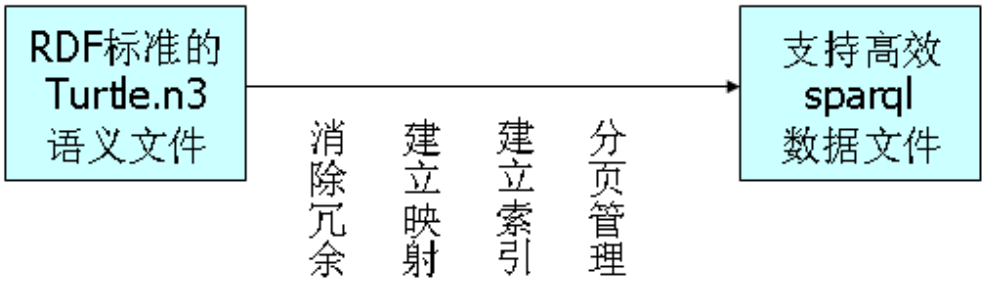


图 2-1　RDF 存储引擎总体需求

2.2 传统 Turtle 格式的数据存在的问题

传统的符合 RDF 标准的 Turtle 格式的数据的表示形式是“主语-谓语-宾语”，这种形式的数据具有以下一些特点：

1、数据冗余。不同三元组的谓语或者宾语相同，甚至谓语和宾语两者都相同，比如在描述人的出生地时可能会出现“张三-出生地-天津.”、“李四-出生地-天津.”这两个句子的谓语和宾语部分相同，这样就造成了数据冗余。虽然 Turtle 格式在存储数据时相同主语的“谓语-宾语”列表可以省略掉第一个主语之后的所有主语，相同主语、谓语的宾语列表可以省略掉第一个“主语-谓语”之后的主语、谓语，但是这种省略的要求很严格，必须是主语或者主语-谓语相同，高效查询需要更为普遍的消除冗余数据的方法。冗余数据给高效查询带来的困难显而易见。

2、缺乏统一排序和索引。这里所指的凌乱是指“主语-谓语-宾语”这样的语句没有高效的排序，虽然 Turtle 格式的数据略写方式要求相同主语的语句放在一起，相同主语谓语的语句放在一起，但是也没有严格规定必须这样做。

3、字符串匹配。Turtle 格式的数据是字符串形式的，字符串的匹配是比较慢的。

4、以 N3 文件形式存储数据。N3 形式的数据实际上还是文本格式的数据，如果有多个文本的话，数据就显得很凌乱，文件之间没有必然联系，信息的组织

比较随机，这也给高效查询带来很大的困难。

2.3 RDF 存储引擎的需求

为了解决传统 RDF 数据存储方式所存在的以上问题，语义数据支持高效的查询，这需要彻底改变语义数据的存储方式，把原始的 turtle 文件消除冗余、建立映射表、建立索引、把索引和数据分页管理。RDF 存储引擎主要有以下几个模块。

2.3.1 Turtle 文件的解析

输入：符合 RDF 标准的 turtle 语义文件。

输出：rawString，是元素的集合，元素的形式是：intLengh-String-intId。

rawFacts，fact 的集合，fact 形式是：intSubject-intPredicate-intObject，

实现一个词法-语法分析器，把输入的 Turtle 文件解析成三元组“主语-谓语-宾语”的语句集合，该集合就是这部分的输出，“主语-谓语-宾语”这样的语句称为事实（fact）。

该部分主要需求是完成词法分析和语法分析。

1）词法分析

的主要任务是解析 Turtle 文件的词语，这些词语包括字符串（包括用三个引号标注的长字符串）、URL、数字（包括 Integer、Decimal）、标点符号、特殊字符（a、True、False、prefix 等）。

2）语法分析

主要任务有以下几点：解析 Turtle 文件开始时前缀申明，解析文件中由前缀修饰的短语，解析 fact 在相同主语或者相同主语-谓语情况下缩写的 fact,解析空节点。

Turtle 文件的解析是整个 RDF 存储引擎的基础，词法-语法解析是数据转换的第一步。

2.3.2 映射表的建立

输入：解析阶段的输出文件 rawString、rawFacts。

输出：StringTable，元素集合，元素的形式是 intId-intLength-String。已经消除 String 的冗余。

Facts 文件，已排序的 fact 的集合，其中每个 Id 都对应 StringTable 中的 intId。

映射表的建立的主要作用是消除数据冗余、避免查找时进行字符串匹配、将所有 fact 进行排序。简单地说，该部分把 Turtle 文件中的 fact 转化为了

“idS-idP-idO”，建立了映射表“id-String”，其中 idS、idP、idO 都是 id 的子集，而 String 中元素字符串互异，并且 id 和 String 之间是一一映射关系，这样消除数据冗余。另外，形式为“idS-idP-idO”的所有 fact 经过严格的排序，首先按照 idS 排序，再按照 idP 排序、最后按照 idO 排序、经过排序的 fact 语句对于索引的建立和压缩具有非常重要的意义，这对查询性能的提高非常关键。在执行查询时都是通过 id 和 hash 值查询，只是把最后的查询结果通过映射表映射为 String，这样就避免了字符串匹配。

2.3.3 索引的建立与压缩

输入：Facts 文件。

输出：15 种压缩后的索引。

根据语义数据的格式特点，建立 15 个索引，即 SPO、SOP、PSO、POS、OSP、OPS、SP、PS、SO、OS、PO、OP、S、P、O。压缩的依据是每个 fact 中各个主语、谓语、宾语之间的差值小。

2.3.4 数据文件生成

输入：15 种压缩后的索引、映射表

输出：数据文件

数据文件依次包含目录页、15 种索引、映射表。每一个部分都按照特定的方式组织页面。

目录页：包含整个数据文件的目录，存储了每个部分的开始页码和结束页码。

15 种索引：每种索引的内部以 B+树的结构组织页面。

原始数据部分：顺序存储原始字符串。

2.3.5 内存排序工具

数据转换过程中需要进行多次排序。内存排序工具实现排序的模板。需要特定的代理函数把内存划分为排序单元，该工具不规定具体的排序规则，需要代理函数作为参数定义其排序规则。需要排序的数据以文件的方式输入，所以排序之前需要调用相应方法把文件映射到内存。

输入：需要排序的文件、解析内存的方法、排序规则。

输出：排序后的文件。

排序的文件大小可能会很大，排序工具需要支持外排序。

2.3.6 文件、内存访问工具

该工具支持对 int 进行编码解码，对 int、string 的特殊读写与跳过。封装在读写内存、文件时的缓冲区的管理。

2.3.7 文件-内存映射工具

该工具调用 windows 接口把文件映射到内存，使文件的变换、排序等操作更加方便。

第三章　RDF 存储引擎的设计

3.1 RDFStore

RDF 存储引擎是 RDFStore 的一个部分,它的作用是把符合 RDF 标准 Turtle 格式的数据经过一定方式的转化，使 sparql 查询语句能在转化后的数据上高效地执行。RDFStore 是一个完整的 RDF 数据管理软件，图 3-1 给出了它的体系结构。

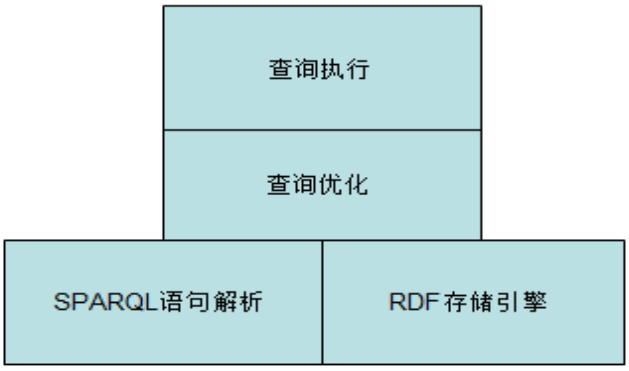


图 3-1　RDFStore 的体系结构

接下来可以对图 3-1 多做点解说

RDF 查询引擎主要包括以下几个部分：

RDF 存储引擎：

RDF 存储引擎的作用是转换 Turtle 格式的数据。将数据排序、建立索引和映射表、建建立文件数据库。

sparql 语句解析部分：

该部分解析 sparql 查询语句。

查询优化部分：

该部分实现查询优化。

查询执行部分：

该部分执行查询语句、输出最后结果。

3.2 RDF 存储引擎的模块设计

3.2.1 RDF 存储引擎的组成

RDF 存储引擎的开发平台是 Windows64 位机。开发工具是 visual studio 2008，开发语言是 C#。图 3-2 给出了 RDF 存储引擎的主要模块结构。

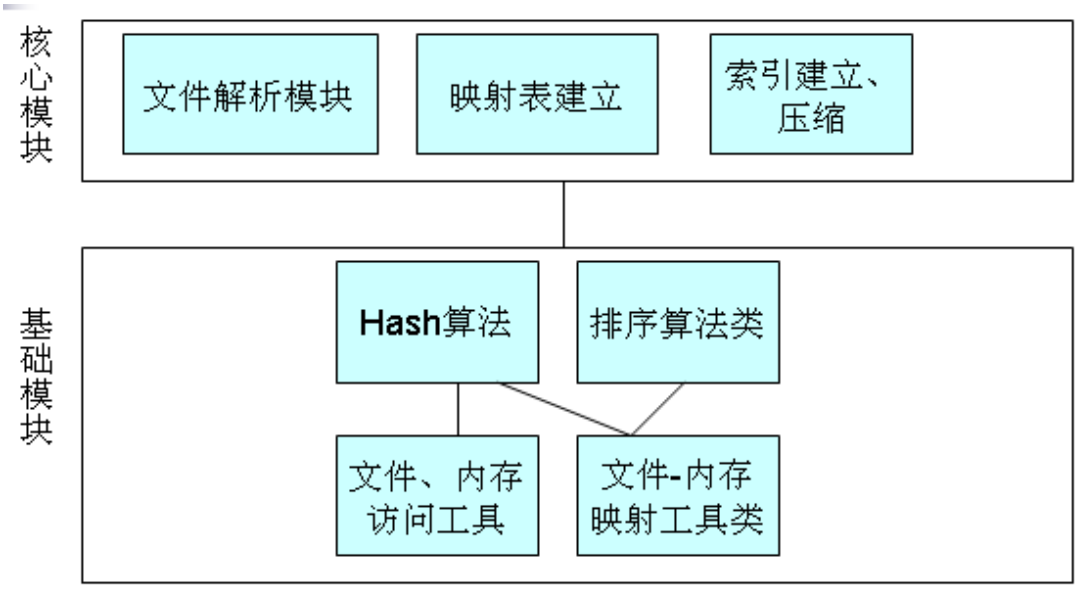


图 3-2　　RDF 存储引擎的模块结构

3.2.2 数据转换流程设计

图 3-3、3-4 展示了数据转换的流程。

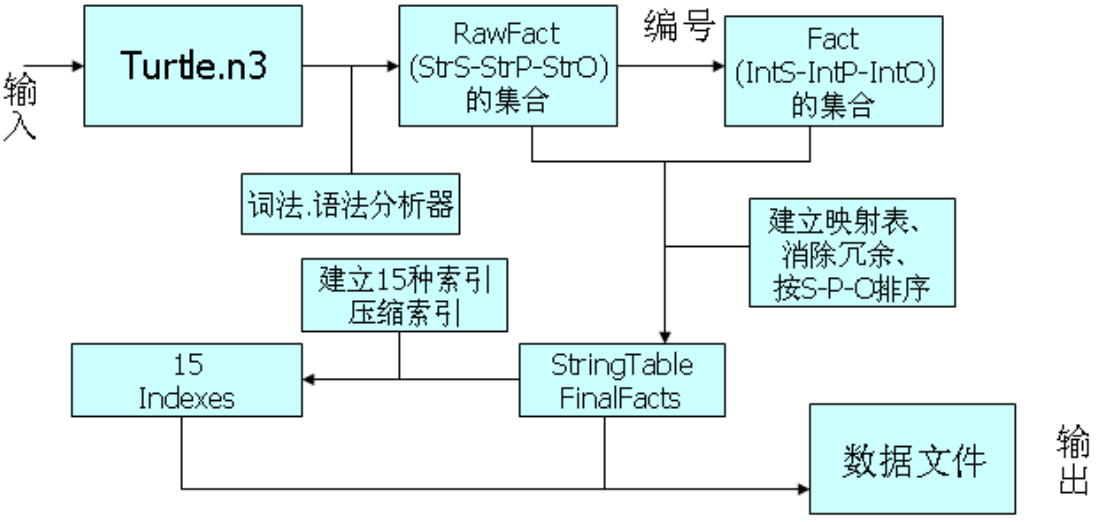


图 3-3　数据转换流程设计

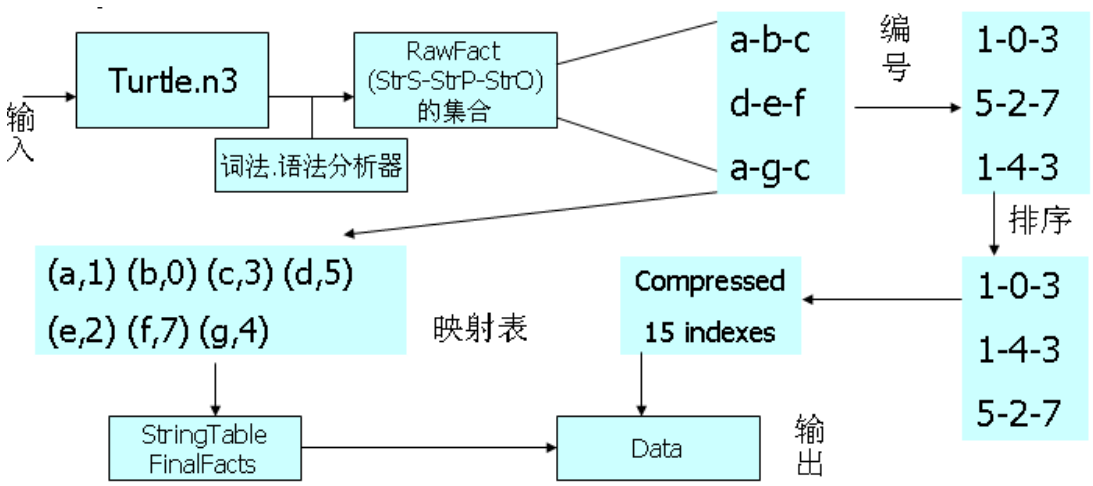


图 3-4　数据转换示意图

3.2.3　基础模块

1）Hash 算法：

提供字符串到 int 类型的 hash 算法，算法的要求是 hash 冲突小。

2）文件、内存访问工具类：

提供文件流、内存的访问方法，实现了 int 和 String 类型在文件、内存中的读写方法和编码方案。

3）排序算法类：

提供了内存中特定的一些字节为单位的排序方法模板，该排序方法为外排序、堆排序，具体的排序规则由具体的 C#代理实现。

图 3-5 展示了外排序的设计。

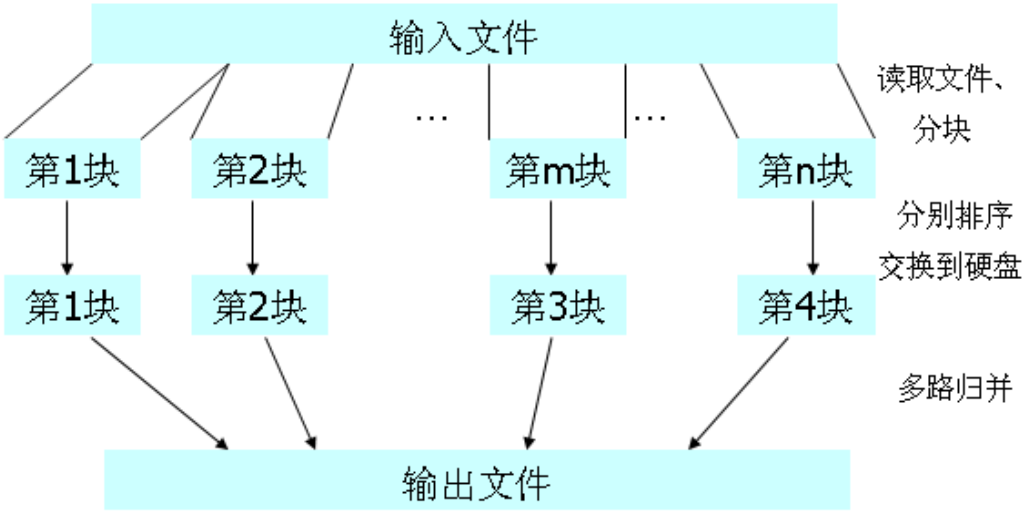


图 3-5　外排序设计

如图所示，读取文件的时候以输入的托管函数的要求将输入文件划分为基本的元素，根据内存的大小定义每块的大小，分块的块数由输入文件和块大小的商决定，每块内部分别从小到大排序，最后实现多路归并。每块内部排序和多路归并的排序规则由输入的托管函数决定。

4）文件内存映射类：

该类调用 windows 平台中的 CreateFile、CreateFileMapping、GetFileSize、MapViewOfFile 等接口将文件映射到内存，以方便数据转换过程中的中间文件的排序和其他变换。

3.2.4 核心模块

1）文件解析模块：

包括一个词法解析器和一个语法解析器、负责解析 Turtle 文件。

词法分析的记号设计：把 Turtle 文件中的记号分为以下几种，Dot,Colon,Comma,Semicolon,LBracket,RBracket,LParen,RParen,At,Type,Integer,Decimal,Double,Name,A,True,False,String,URL。

词法解析器输入缓冲的设计：使用 C#设计语言的输入流进行读取字符，这里使用超前读入一个字符的方法识别记号。

词法解析器错误处理方案设计：采用“紧急方式”错误处理，即反复删除掉剩余输入最前面的字符，直到词法分析器发现一个正确的记号为止。

词法分析器的其他功能设计：该词法分析器能够过滤掉文件中的注释行、空格、制表符、换行符。

2）映射表建立模块：

此模块的作用是对字符串进行编号，消除重复冗余数据，建立编号 id 到实际字符串的映射表。

编号的设计通过 hash 值的计算初步消除冗余，

3）索引建立、压缩模块：

该模块的作用是建立 15 个索引，并且定义了各种索引压缩的算法和索引的存储方式。为了避免字符串的匹配，先对 fact 中的每一个字符串进行编号，主语和宾语的编号为奇数，从 1 开始递增，谓语的编号为偶数，从 0 开始递增。对于每一个字符串，通过 hash 算法确定是否需要重新编号，若该 hash 值存在且编号奇偶性相同，则不需要编号而直接返回已有编号。再经过一些方法，消除映射表“id2-String”中相同的字符串。建立这样一个映射表，从“id1-id2”映射到“id2-String”，其中 String 是 Turtle 中字符串的集合，id1 是一个映射前 id 的列表，id2 是映射之后 id 的集合，每一个 id2 中的元素对应一个 String 中的元素，String 元素之间是没

有重复的。id1 是 fact 映射之前的主语、谓语、宾语、之一，不同的 id1 之间可能对应相同的 id2., 经过映射之后的 fact 主语、谓语、宾语的 id 都在集合 id2 中，每一个 id2 中的元素都对应不同的字符串，这样就消除了数据的冗余。通过 hash 值和 ID 值来查询也避免了字符串的匹配。

4）文件数据库生成模块：

该模块负责生成最后的文件数据库，定义了文件数据库分页的规则以及每页的内容格式。

数据文件的格式如图 3-6 所示。

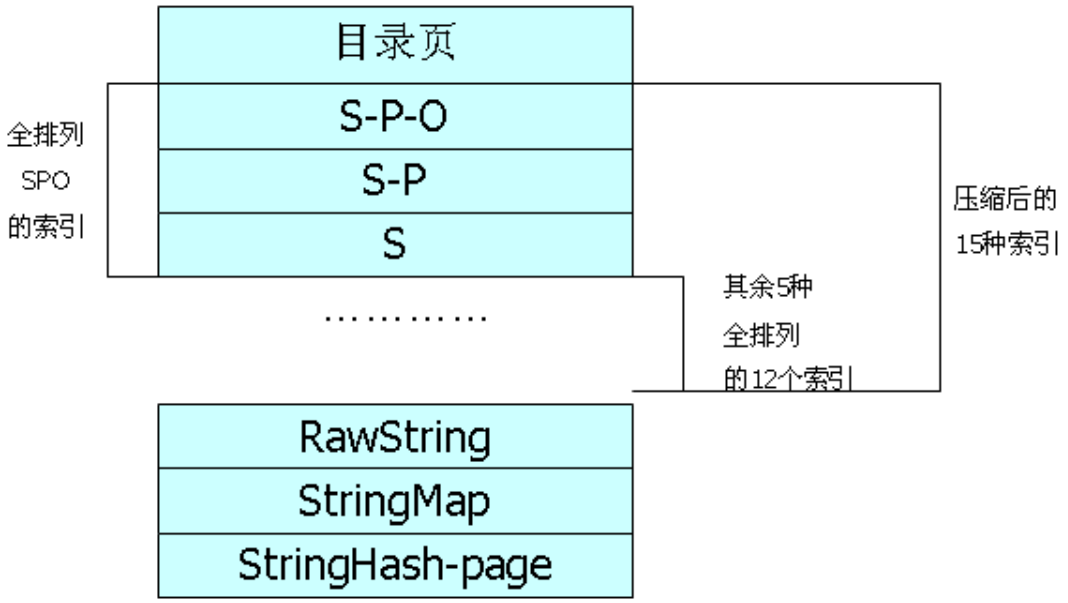
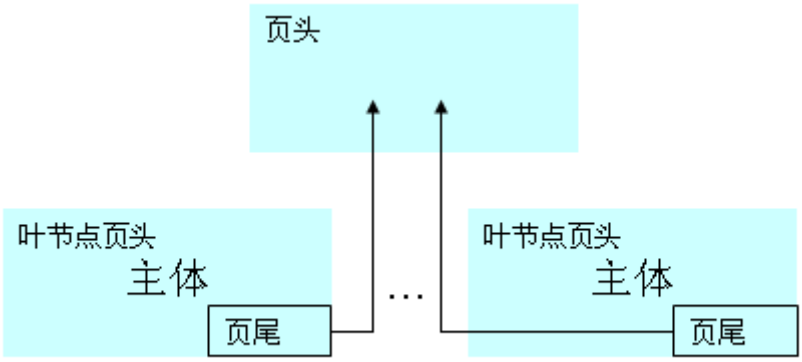


图 3-6　数据文件格式

- 索引的存储管理

采用 B+树管理索引页。由于索引的主语、谓语、宾语经过严格的排序，索引使用每个索引页的最后一个索引作为该页的标志，将该页的标志存储在其父节点中。如图 3-7 分页管理。



3-7　分页管理

叶子节点的页头设计：页头为 4 个字节，记录下一个叶子节点的页码数，如果为该部分最后一页则全为 0。

非叶子节点页头设计：页头 16 个字节，头 4 个字节全 1，气候 4 个字节记录下一页的页号，如果为该部分的最后一页则全 0，其后 4 个字节记录该页包含下级页面的个数。

非叶子节点主体设计：记录每页的标志和页号。

- 原始字符串的存储管理

采用顺序存储管理页。

页头设计：页头为 8 个字节，前 4 个字节记录下一页的页码。若为最后一页则全为 0，后 4 个字节记录当前页所包含的原始字符串的个数。

页主体设计：每个字符串的格式是“intId-intHash-intLenth-String”，排列顺序是按照 intId 从小到大排序。

第四章　RDF 存储引擎的实现

4.1 基础工具模块

4.1.1 Hash 算法

提供字符串到 int 类型的 hash 算法，算法的要求是 hash 冲突小。

此工具类的作用是把任意长字符串作为输入，将其进行运算输出一个 4 字节 int 类型的整数，hash 算法的要求是 Hash 冲突小。该工具提供了多种方法以适应不同的输入字符串形式。第一种是以字符串作为输入字符串，第二种是以字节数组作为输入字符串，第三种是以一个 Byte 指针作为字符串的开始、以一个 int 参数指定字符串的长度确定一个数组。

4.1.2 文件内存访问工具类

此工具类提供文件流、内存的访问方法，实现了 int 和 String 类型在文件、内存中的读写方法和编码方案。重要的方法主要有以下几个：

1、Write 方法：

该方法有两个参数，一个为 Byte 指针，一个 int。作用是将从指定内存位置开始的整数个字节的内容输入到指定文件。

2、WriteId 方法：

该方法作用是把一个整数按照一定的编码方式输出到指定文件。编码方法如下：

```
while (id >= 128)
        {
           Write((Byte)(id | 128));
          id >>= 7;
       }
Write((Byte)id);
```

编码时以每个 Byte 的最高位作为标识符。如果为 1，表示该 int 还没有结束，如果为 0，表示该字节已经结束。把 int 输出到文件时先写低位再写高位，每 7 位写入到一个字节。WriteId 方法主要用于写字符串的长度和字符串的 id。

3、WriteString 方法：

该方法的作用是写字符串到指定输出文件。写字符串的时候总是先写字符串的长度，再写实际字符串。写字符串的长度是调用 WriteId 方法，写实际字符串的时候调用 Write 方法。

4、SkipId、SkipString方法：

这两个方法作用是在从内存读取数据时用来跳过 Int、String。SkipId 作用是跳过以 WriteId 方式编码的 int 整数。返回跳过之后的下一个 Byte 指针。SkipString 作用是跳过字符串长度和实际字符串，返回跳过当前内容的下一个 Byte 指针。

5、ReadId方法：

读取当前的 int 数，返回读取当前 int 数之后的下一个 Byte 指针，所读取的值以引用参数的形式返回。

6、ReadString方法：

读取当前字符串。先读取字符串的长度以引用参数的形式返回，字符串的实际开始位置以 Byte 指针类型引用参数的形式返回。返回读取 String 之后的下一个 Byte 指针。

4.1.3 排序算法类.

此类提供了内存中特定单元为单位的排序方法模板，该排序方法为外排序、堆排序，具体的排序规则由具体的 C#代理实现。

该类的作用是将内存重新排序。该方法的输入是一个文件、一个定义解析元素单元的 dSkip 托管函数、一个定义比较规则的 theSorter 托管函数。该工具通过特定接口把文件映射到内存，按照特定的 dSkip 规则把文件解析成若干个元素，在按照排序规则 theSorter 把内存重新排序。最后把结果输出到一个文件,该工具使用堆排序，由于需要排序的文件很大，该工具使用外排序。

该排序算法通过传入的 dSkip 托管函数把映射到内存的文件解析成结构体 Range，然后把 Range 进行排序。Range 有两个 Byte 指针成员，分别表示 Range 的开始地址和结束地址。

Range 的定义如下：

```
struct Range
        {
            public Byte* from;
            public Byte* to;
           …
}
```

该工具一个重要的方法就是 Sort 方法，该方法的形式如下：

sort(ref TempFile in,ref TempFile out, dSkip skip,theSort comp)

托管函数参数 dSkip 的形式如下：

delegate Byte* dSkip(Byte* b1);

skip 主要定义把输入文件解析为结构体 Range 的方法。

托管函数参数 theSorter 的形式如下：

delegate Int32 theSorter(Byte* b1,Byte* b2);

theSorter 主要定义 Range 排序的规则。

4.1.4 文件内存映射类

该类调用 windows 平台中的 CreateFile、CreateFileMapping、GetFileSize、MapViewOfFile 等接口将文件映射到内存，以方便数据转换过程中的中间文件的排序和其他变换。

4.2 RDF 数据文件解析模块

4.2.1 RDF 数据文件解析过程

这是数据转换的第一步工作，该模块的作用是读入符合 RDF 标准的 Turtle 格式数据文件，将其进行词法分析、语法分析，最终输出 RDF 文件中所有“主-谓-宾”格式的语句，这样的语句称作 fact,fact 中的主、谓、宾 3 个部分都是由原始 Turtle 文件经过解析模块的解析后字符串再经过特定 Hash 算法生成的 Int 类型的数字。此后每一个主语或者谓语或者宾语称为一个元素。文件解析阶段将输出两个临时文件：rawFacts、rawString。

rawFacts:

fact 的集合，fact 形式是：intSubject-intPredicate-intObject，fact 出现的顺序和原始 Turtle 文件中 fact 的顺序一致，主谓宾各部分都是由原始 Turtle 文件的主谓宾字符串通过特定的算法得到的编号 id。

rawString:

是元素的集合，元素的形式是：intLengh-String-intId。其中 intLength 是指字符串的长度，单位为 byte，字符串中每个字符用 Unicode 表示，因此每个字符占用两个字节；String 是从 turtle 文件解析出来的原始字符串；intId 是每个元素的编号，主语和宾语的元素编号为基数从 1 开始递增，谓语的元素编号为偶数，从 0 开始递增。所有元素的顺序和原始 turtle 文件元素的顺序一样，相同主语的“谓语-宾语”短语省略掉主语的 fact 补齐为主谓宾短语，相同“主语-谓语”的宾语省略掉“主语-谓语”短语的 fact

也补齐为主谓宾短语。

文件解析模块部分主要包括两个部分，词法分析部分和语法分析部分。

4.2.2 词法分析

词法分析部分的输入是 RDF 格式的 Turtle 文件，输出的是以下几种词：

Dot,Colon,Comma,Semicolon,LBracket,RBracket,LParen,RParen,At,Type,Integer,Decimal,Double,Name,A,True,False,String,URL。

Dot：表示点号“.”。文件中可能是小数点和语句结束符号。

Colon：表示冒号“:”。用分隔元素的修饰前缀和被修饰部分，在空节点的表示中也会用到。

Comma：表示逗号“,”。用来分隔因为“主语-谓语”相同的省略掉“主语-谓语”的宾语列表。

Simecolon：表示分号“;”。用来分隔因为主语相同而省略掉主语的“谓语-宾语”列表

LBracket：表示左方括号“[”。用来表示空节点。

RBracket：表示右方括号“]”。用来表示空节点。

LParen：左小括号“(”。用来表示空节点

RParen：右小括号“)”。用来表示空节点

At：表示符号“@”

Type：表示类型

Integer：整数。

Decimal：表示小数,如：5.8。

Double：表示带有“E”或者“e”表示次方的小数，如：6.2e12 , 72E-3

Name：名字,没有用双引号引住的普通字符串。

A：字符串“a”。

True：字符串“True”。

False：字符串“False”。

String ：用双引号引住的普通字符串。

URL：用“<”和“>”括住的普通字符串。

词法分析的主要过程如图 4-1 词法分析.

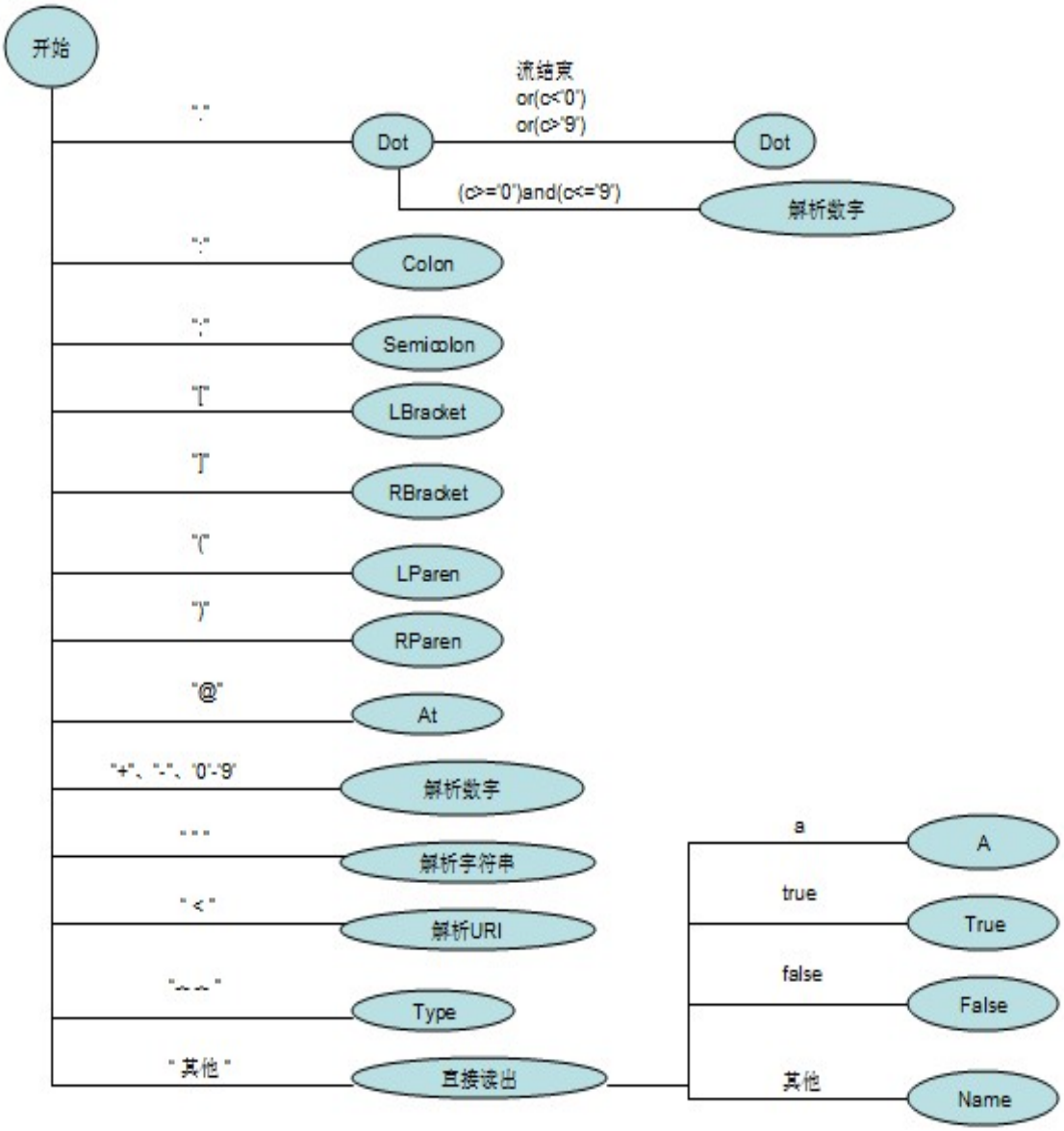


图 4-1：词法分析器

字符串的解析就是解析出引号之间的部分。

URL 解析就是解析出“<”和“>”之间的部分。

解析数字的过程如图 4-2：

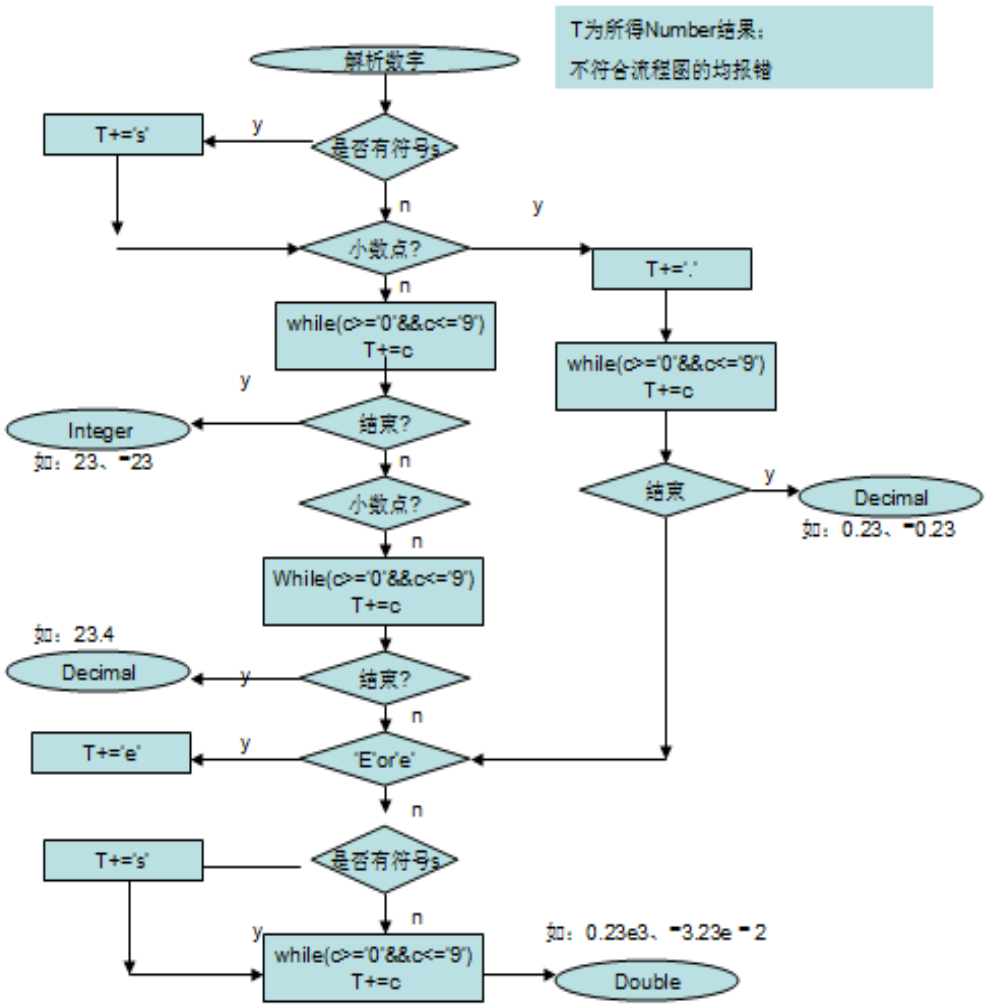


图 4-2：数字解析流程

4.2.3 语法分析

RDF 格式数据的标准语法典型格式是“主语-谓语-宾语”的三元组，整体结构有以下几种情况：

1）主语 谓语 宾语.

2) 主语 谓语 宾语;谓语 宾语;谓语 宾语.

相同主语的谓语-宾语列表之间用分号“;”隔开。

3) 主语 谓语 宾语;谓语 宾语,宾语.

相同主语-谓语的宾语列表之间用逗号“,”隔开。

基于以上情况，语法分析的总体过程如图 4-3:

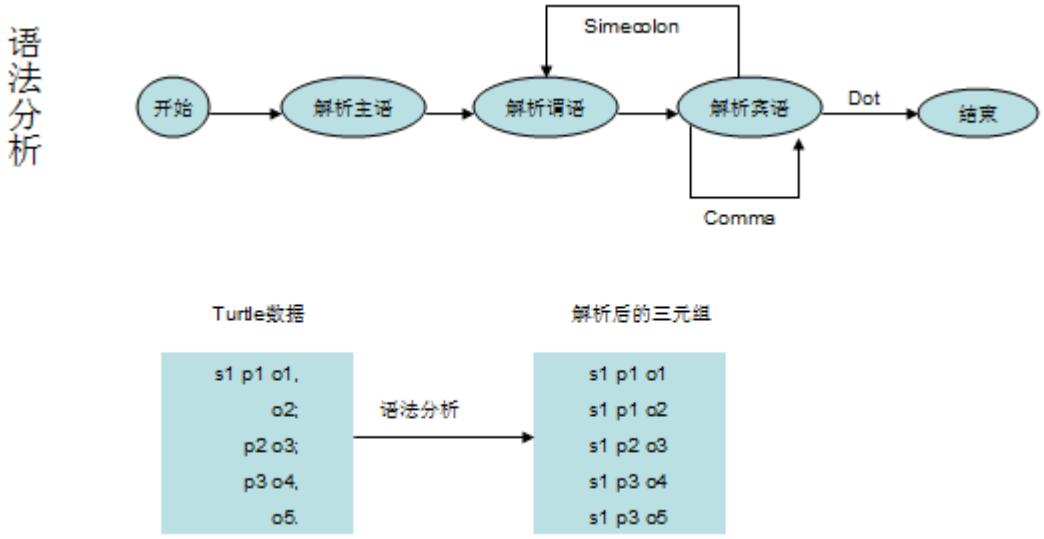


图 4-3　语法分析

解析主语、谓语、宾语的过程差不多，这里只展示主语的解析过程，如图 4-4：

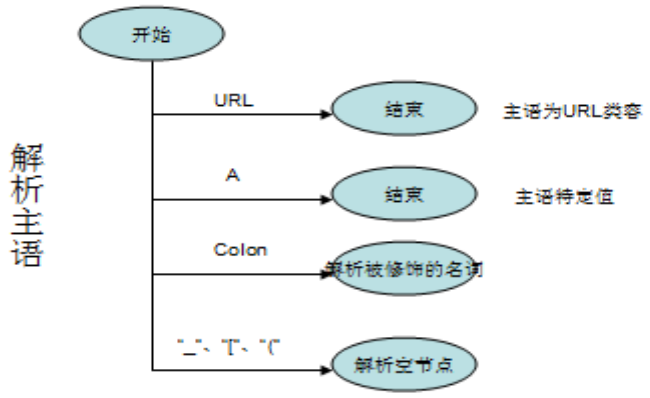


图 4-4　解析主语

在开始解析 Turtle 文件的时候，需要解析文件头，文件头中定义了文件中出现的前缀，用这些前缀来修饰其后面的词语。

文件头的解析：

文件头前缀有以下几种情况:

1) @ base : URL

2) @ prefix prefixname : URL

被修饰的词解析:

被修饰的词形式为“prefixname : realname”，被解析后的格式为

“URLrealname”，其中的 URL 部分为文件头解析时 prefix 部分对应 prefixname 的 URL。

4.3 RDF 映射表建立模块

4.3.1 RDF 映射表

由于 RDF 文件解析模块得到的 rawFacts 是由原始的符合 RDF 标准 Turtle 格式的文本解析而来，Turtle 格式的文件有很多相同的字符串。比如： s1 p1 o1,s1 p1 o2 两个 fact 当中的 s1 和 p1 是重复的。这样会造成大量的冗余信息，不方便存储，而且降低查询效率。

RDF 映射表的建立目标是生成一个 StringTable 和 facts 文件，该部分还有一个重要的中间临时文件 StringIds。

StringTable：

由元素排列形成，元素的形式是 intId-intLength-String。其中 String 包含所有 fact 中的字符串，而且不重复；intId 是字符串的编号，用以映射时通过 id 查找相应的字符串；intLength 是字符串的长度。所有元素按照一定顺序排列。

Facts 文件：

原始 RDF 经过解析后的 rawFacts 在经过主语、谓语、宾语排序后的 TempFacts，Facts 的元素形式是: idSubject-idPredicate-idObject，其中每个 Id 都对应 StringTable 中的 intId，而且每个 id 都是由 TempFacts 经过 StringIds 映射后得到的。

StringIds 文件：

文件是一个有许多元素组成的列表，元素的形式是 intId1-intId2,其中 intId1 对应 RDF 文件解析模块中生成 RawFacts 中的每个元素的 intSubject、ntPredicate 或者 intObject，intId2 对应 StringTable 中的 intId 部分，此文件的作用就是把 rawFacts 中的主语谓语宾语映射到消除冗余字符串后的 StringTable 上。

4.3.2 RDF 映射表建立步骤

该模块把 RDF 文件解析模块的输出 rawString 作为输入，rawString 是元素的集合，元素的形式是：intLength-String-intId。该部分的输出文件有三个 StringTable、StringIds 和 facts。

stringIds 的生成主要经过以下几个步骤：

1） 排序：把 rawString 中的元素进行排序，排序规则是按照 String 部分排序，若 String 部分相同则按照 intId 部分排序，在按照 intId 排序时，规则是偶数小于基数，intId 为偶数的元素排在前面，该步骤得到的结果为 sortedString，其中每个元素的形式与 rawString 的形式相同，为 intLength-String-intId。

2） 生成 rawIdMap：通过 sortedString 生成 rawIdMap，rawIdMap 元素的形式为 IntId1-IntId2。

rawIdMap 的生成算法：

```
Int lastId,id;
 String lastString, nowString;
while(getNext element in sortedString)
{
   Id=element.id;
   nowString=element.string;
   if(nowString equals lastString)
 {
 rawIdMap.write(lastId);
 rwaIdMap.write(id);
 }
 else
 {
     rawIdMap.write(id);
     rawIdMap.write(id);
      lastId=id;
     lastString=nowString;
 }
}
```

3） 排序，生成 idMap：把 rawIdMap 的元素 IntId1-IntId2 按照 IntId1 进行排序，其中 Int 的排序规则是偶数小于基数，偶数、基数内部按照整数大小比较规则，所有元素从小到大排列。

4）newId 的生成：newId 的元素的形式为 IntId1-IntId2，根据 idMap 生成 newId 的算法如下：

```
int firstId,currenId,newId,lastId;
newId=0;
While(getNext element in idMap)
{
     currentId=element.secendId;
     if(firstId!=lastId)
     {
          ++newId;
            lastId=firstId;
```

```
            }
            newIds.writeId(currentId);
            newIds.writeId(newId);
        }
```

5) 排序：把 4）中生成的 newId 排序，规则是按照 intId1 排序，按照整数大小从小到大排列（此处不分奇数偶数）。最后得到 stringIds，这就是映射表建立模块的输出文件之一。

StringTable 的生成步骤：

6） 变换元素形式、消除重复的字符串：

把 sortedString 的元素形式由 intLengh-String-intId 改变为 intId-intLengh-String，结果为 stringList。

由于 sortedString 是按照 String 部分排序的，所以 String 相同的 sortedString 元素会在一起，一次读入每个元素，若 String 不分与前一个元素的 String 部分相同，这个元素不被写入 stringList，从而消除的重复的字符串。

7） 排序生成 StringTable：

由 stringList 生成 StringTable，把 stringList 中的元素按照 id 进行排序，排序规则是偶数小于基数，偶数基数内部，按照整数的比较大小规则排序，整个 StringTable 元素从小到大排列。

Facts 文件的生成过程：

1) 按主语排序：

由 RawFacts 按照 idSubject 排序，得到 sorteBySubject。

2） 映射主语、变换元素：

把 sortedBySubject 由 intSubject-intPredicate-intObject 转换为 intPredicate-intObject-intSubResolved,其中 intSubResolved 由 intSubject 经过 StringIds 解析后得到。

3） 按谓语排序：

得到 sortedByPredicate，元素形式为：intPredicate-intObject-intSubResolved

4） 映射谓语、变换元素：

得到形式为 intObject-intSubResolved-intPreResoved 的列表

5） 按宾语排序：

得到 sortedByObject,其元素形式为：

intObject-intSubResolved-intPreResoved

6） 映射宾语、变换元素：

得到形式为 intSubResolved-intPreResoved-intObjResolved 的列表。

7） 排序：

把 6）步骤中得到的列表排序得到最终的 facts 文件。文件元素形式为：

intSubResolved-intPreResoved- intObjResolved，排序规则是先按照 intSubResolved 排序，再按照 intPreResoved 排序，最后按照 intObjResolved 排序。

4.4 RDF 索引压缩算法

4.4.1 RDF 索引压缩原理

RDF 数据存储引擎需要再 RDF 映射表建立模块的输出文件 facts 上建立 15 个索引。S 代表主语，P 代表谓语，O 代表宾语，N 代表一个整数用以表示数目。

15 个索引分别是：

第一类：SPO、SOP、PSO、POS、OSP、OPS

第二类：SP、PS、SO、OS、PO、OP

第三类：S、P、O

由于索引的文件比较大，所以需要进行压缩。

压缩原理，这 15 索引都是由 RDF 映射表建立模块的输出文件 facts 计算而来，由于 facts 文件是按照 SPO 排序得到。相同主语、不同谓语的 fact 会排在一起；相同主语谓语、不同宾语的 fact 也会排列在一起。相邻两个 fact 若主语不同，则两个主语数值相差不大；相邻两个 fact 若主语相同、谓语不同，则谓语的差值也不大；同理，相邻两个 fact 若主语谓语都相同、宾语不同，则宾语差值也相对不大。

4.4.2 RDF 索引压缩算法

本系统将生成的索引以分页的方用文件式存储起来。分页的规则如下：

每页的大小可以设定，此处使用 1384，单位为字节。每类压缩后的索引都有特殊的页头，页头后才是索引，一个 SPO 或者 SP 或者 S 的索引称为一个索引单位。每页的地第一个索引单位不进行压缩，直接写入原始索引单位。每页的最后一个索引单位作为页边界单独用 boundary 存储起来，便于搜索。

15 个索引分为 3 类下面详细讨论每类索引的压缩方法。

第一类索引压缩以 SPO 为示例：

依次读入每个索引单位，当前索引单位为 S-P-O，前一个索引单位为 LS-LP-LO。

```
if(S==LS)
{
    If(P=LP)
    {
        If(O==LO)
        {continue;}
        Else
```

```
            {
                if ((O – LO) < 128)
                    {
                //若(S=LS)&&(P=LP)&&(O-LO<128)  直接写入O-LO的值
                        Write (O – LO));
                    }
                     else
                     {
                   //若(S=LS)&&(P=LP)&&(O-LO>128)
                   //写入索引头表示O-LO的长度，再写入O-LO的值
                    Write((Byte)(0x80|(bytes(O – LO -128)-1)));
                        writeDelta(O – LO - 128);
                     }
            }
        }
        Else
        {
        //若主语相同、谓语不同
        //写入一个字节的索引单位头，表示 P-LP 差值的长度、宾语的长度
        //写入 P-LP 的值，写入完整宾语 O
          Write((Byte)(0x80|(bytes(P-LP)<<2)|(bytes(O)-1)));
            writeDelta(P-LP);
          writeDelta(O);
        }
    }
    Else
    {
    //若主语不同写入一个字节表示主语、谓语、宾语各自所占字节数，写入
    完整主语 S、谓语 P、宾语 O
    Write((Byte)(0xC0|((bytes(S-LS)-1)<<4)|((bytes(P)-1)<<2)|(bytes(O)-1)));
    writeDelta(S-LS);
    writeDelta(P);
    writeDelta(O);
    }
```

在查询的时候先读一个索引单位的第一个字节，若最高位为 1，则此字节为该索引单位的索引头，该字节表示了该索引单位的长度；若最高为 0，则此字节就表示一个索引单位，该索引单位的主语谓语与前一个索引单位相同，该字节的值表示当前索引单位宾语与前一个索引单位的差值，该差值小于 128。

第二类索引压缩以 SP 为示例：

依次读入每个索引单位，当前索引单位为 S-P，前一个索引单位为 LS-LP。C 代表计数器。

```
    if（ （S==LS）&&(P – LP< 32 )&&（ C < 5））)
    {
```

```
        //此时没有索引单元头
        write( (C - 1) * 32 + (P-LP))
    }
    else
    {
        //索引单元头，表示（S-LS）的长度、P的长度、C的长度
        Write（(Byte)((0x80) | (bytes0(S – LS) * 25 + bytes0(P) * 5 + bytes0(C -
    1)))）;
        //写（S-LS）、P、C
        writeDelta0(S-LS);
         writeDelta0(P);
        writeDelta0(C-1);
    }
```

如果满足条件（S==LS）&&(P – LP< 32)&&（ C < 5），则该索引单元只有一个字节，该字节最高位是 0，接下来两位可以表示 0-3 四个数，由于 C 不会为 0，所以将其加 1，表示 1-4 之间的一个数，低 5 位可以是 0-31，将其加 1 表示 1-32 之间的一个数。

如果不满足条件则，最高位是 1，表示第一个字节是索引单元头。

同样，在查询的时候根据每一个索引单元的第一个字节的最高位来确定是否有索引单元头，若最高位为 1，则为索引单元头，可根据该索引单元头计算该索引单元每个部分的长度。

第三类索引以 S 为例：

依次读入每个索引单位，当前索引单位为 S-P，前一个索引单位为 LS-LP。C 代表计数器。

```
    if (((S-LS) < 16) && (C<= 8))
    {
        write（(Byte)(((C - 1) << 4) | (nowSubject - lastSubject)));
    }
    else
    {
      Write((Byte)(0x80 | ((bytes0(S - LS - 1)) * 5 + bytes0(C - 1))));
      writeDelta0(S-LS);
      bufferPos = writeDelta0(C-1);
      }
```

此处依然用第一字节的最高位标志是否为索引单元头。若为 0，表示该索引单元就是当前字节，最低 1-4 位表示 S-LS，4-7 位表示数目。

4.5 索引、数据文件的生成

4.5.1 索引文件格式

上一节主要介绍了索引压缩算法。这一节主要详细说明被压缩后的索引以及数据的文件生成方法。

本系统将生成的索引以分页的方用文件式存储起来。分页的规则如下：

每页的大小可以设定，此处使用 1384，单位为字节。每类压缩后的索引都有特殊的页头，页头后才是索引，一个 SPO 或者 SPN 或者 SN 的索引称为一个索引单位。每页的地第一个索引单位和最后一个索引单位不进行压缩，直接写入原始索引单位。每页的第一个索引单位和最后一个索引单位作为页边界单独用 boundary 存储起来，便于搜索。

S、P、O 的全排列由 6 种。15 种索引分别是：SPO、SOP、PSO、POS、OSP、OPS、SP、PS、SO、OS、PO、OP、S、P、O。

把这最后的索引文件的生成顺序与 SPO 全排列的顺序有关，谓语索引的类别无关。现在把这 15 种索引按照 6 种全排列分为 6 组，并对每小组进行编号，编号从 0 开始，依次递增。

0 组） 顺序：SPO； 成员：SPO、SP、S

1 组） 顺序：SOP； 成员：SOP、SO、S

2 组） 顺序：OPS； 成员：OPS、OP、O

3 组） 顺序：OSP； 成员：OSP、OS、O

4 组） 顺序：PSO； 成员：PSO、PS、P

5 组） 顺序：POS； 成员：POS、PO、P

注意到第 2n 组和第 2n+1 组的最后一个成员相同。这里把这个成员划分给前者。所以最后的分组情况是：

0 组） 顺序：SPO； 成员：SPO、SP、S

1 组） 顺序：SOP； 成员：SOP、SO

2 组） 顺序：OPS； 成员：OPS、OP、O

3 组） 顺序：OSP； 成员：OSP、OS

4 组） 顺序：PSO； 成员：PSO、PS、P

5 组） 顺序：POS； 成员：POS、PO

所以最后在压缩后的索引文件中出现的索引顺序是按照组号出现的，每小组内部的所引数序和分组中成员出现顺序一样。

Order 表示组号，则总体生成索引文件的方法如下：

```
For(int order=0;order<6;order++)
{
```

```
生成第一类索引（order）;
生成第二类索引（order）;
If(order 为偶数)
 {
  生成第三类索引（order）;
 }
}
```

在建立索引的时候需要把映射表建立阶段的输出文件 facts 中的 fact 读取出来。

这 15 个索引生成的而过程中，每个索引都会在一个新页的开始，每个索引的最后页写不满时用 0 填充，这样每个索引都占完整的整数个页。每个索引生成都要记录其开始的页码和页数，以便查询使用，记录每个索引的开始页码和页数是使用一个结构体实现的。这个结构体的内容将会写到整个数据的最后页。

4.5.2 数据文件生成

本节将对本系统输出文件数据库进行详细的说明。

本文件数据库的分页大小可以设定。该文件数据库讲一次出现以下几个部分：目录页、0-5 号分组索引、原始字符串、映射表。

目录页：

目录页包含了一个结构体，该结构体以字节流的方式存入数据库。目录页详细记录了 15 个索引的起始页码、页码数，原始字符串的起始页码、页码数，映射表的起始页码、页码数。

原始字符串：

原始字符串包含了从 turtle 文件中解析出来的字符串、字符串长度、对应的 hash 值，id 值。

映射表：

映射表包含了 hash 值和与其对应的字符串所在的原始字符串所在的页以及页内的偏移量。

数据库各个部分的每页的也头都记录了关于该页的相关信息。如原始字符串的每页都记录了该页内字符串的个数。

由于各个部分的页码可能很大，所以每个部分页的组织也必须有一定的规律。在每个部分的数据页后面有一页或多页该部分的子目录，这个子目录记录了该部分每个页码的边界元素与页码的对应情况。子目录的他也有文件头，这个文件头的前几位用 1 填充，文件头还记录了该子目录页所记录的边界元素的个数。

4.6 小结

本章主要介绍了 RDF 存储疫情的各个模块的具体实现方法。整个模块和代码文件的对应关系如下：

Hash.cs：hash 算法

Sorter.cs：排序算法类

StringLookup.cs：字符串编号类

TempFile.cs：文件、内存访问工具类

TurtleParser.cs：此法解析器、语法解析器

MemoryMappedFile.cs:调用 Windows 接口，建立文件-内存工具类

rdf3xload.cs：主程序类，包括映射表的生成模块

DatabaseBuilder.cs：索引生成、压缩工具，文件数据库生成工具

第五章　总结和展望

本次毕业设计达到了预期目标，实现 RDF 存储引擎的基本功能，以一种高效的、消除冗余的、经过压缩的、有序的方式对语义数据进行了存储。此次毕业设计历时 6 个月、代码量 3400 行。

RDF 存储引擎吸纳了一些传统数据库的优点对语义数据进行转换压缩，以分页的方式组织数据，建立了适当的索引，使 sparql 查询语句能在大量数据的基础上高效地执行。RDF 存储引擎消除了以传统文本文件存储语义数据的数据冗余、组织形式不支持高效查询等缺点。

本 RDF 存储引擎还有许多需要改进的地方：该存储引擎只支持查询语句，每一次数据的增加、修改、删除都会造成所有数据的重新处理；该系统只能处理 Turtle 形式的语义数据；　该系统只能在 Windows 系统上运行。

随着语义网络的发展，语义数据的数量还将会迅速增长，高效的语义数据管理系统的研究相当重要。现在语义网技术还不是特别成熟，还没有出现出众的、高效的语义数据管理方案。语义数据管理系统的发展还有很长的路程。虽然这样，语义网技术必将会成为将来互联网技术的一个重要分支，语义网技术必将会对未来互联网技术的发展产生重要的影响，而高效语义存储引擎的研究也会变得越来越重要。

参考文献

[1] John Hebeler, Matthew Fisher, Ryan Blace. Semantic Web Programming.[EB/OL]. www.wiley.com/compbooks, 2009.

[2] Thomas Neumann, Gerhard Weikum. RDF-3X: a RISC-style Engine for RDF[R]. JDMR (formely Proc. VLDB). Auckland, New Zealand, 2008.

[3]Thomas Neumann, Gerhard Weikum. Scalable Join Processing on Very Large RDF Graphs[R]. SIGMOD ,Providence, USA,2009.

[4] D. J. Abadi, A. Marcus, S. Madden. Scalable semantic web data management using vertical partitioning[R]. VLDB, Vienna, Austria, 2007.

[5] S. Alexaki et al. The ics-forth rdfsuite: Managing voluminous rdf description bases[R]. SemWeb, 2001.

[6] K. Anyanwu, A. Maduko, A. P. Sheth. Sparq2l:towards support for subgraph extraction queries in rdf databases[R]. WWW, 2007.

[7] S. Auer et al. Dbpedia: A nucleus for a web of open data[R]. ISWC/ASWC, 2007.

[8] L. Baolin,H. Bo. Path queries based rdf index[R]. SKG, 2005.

[9] L. Baolin,H. Bo. Hprd: A high performance rdf database[R]. NPC, 2007.

[10] Thomas Neumann, Guido Moerkotte. A Framework for Reasoning about Share Equivalence and Its Integration into a Plan Generator[R]. BTW . Münster, Germany, 2009.

[11]Thomas Neumann, Sebastian Michel. Algebraic Query Optimization for Distributed Top-k Queries[R].BTW.Aachen, Germany, 2007.

[12]Sven Helmer, Robin Aly, Thomas Neumann, Guido Moerkotte. Indexing Set-Valued Attributes with a Multi-level Extendible Hashing Scheme[R].DEXA,Regensburg, Germany, 2007.

[13]Thomas Neumann, Sebastian Michel. Algebraic Query Optimization for Distributed Top-k Queries[R].IFE, 2007.

[14]Sebastian Michel, Thomas Neumann. Search for the Best but Expect the Worst - Distributed Top-k Queries over Decreasing Aggregated Scores[R].WebDB, Beijing, China, 2007.

[15] Sven Helmer, Robin Aly, Thomas Neumann, Guido Moerkotte. Indexing Set-Valued

Attributes with a Multi-level Extendible Hashing Scheme[R]. DEXA Regensburg, Germany, 2007.

致　　谢

本论文的完成得益于很多人的指导，关心和帮助。

感谢张坤龙老师，给予我论文各个方面极大的指导和帮助。张老师广博的专业知识、严谨的治学态度、为人师表的工作风范使我终生受益。无论是在理论学习阶段，还是在论文的选题、资料查询、开题、研究和撰写的每一个环节，无不得到张坤龙老师的悉心指导和帮助。张老师不仅传给我知识，更重要的是培养了我分析问题和解决问题的能力，使我具备了作为科学工作者的基本素质和技能，为我今后的研究生学习提供基础。在此，衷心感谢毕业设计期间张坤龙老师对我的关心和指导。

感谢计算机科学与技术学院的诸多曾经给我授课和给我指导的老师。没有他们，我不可能具备扎实的基本功和技术功底来完成本论文。他们严谨治学的工作态度和广博深厚的专业知识给我留下了深刻的影响。

感谢我的学长孙博，在学习中给我的帮助和鼓励，并且共同探讨问题，使我受益匪浅。他的聪明智慧也常常给我的学习带来灵感。毕业设计期间我极大得益与于他的交流。

最后衷心感谢我的家人。爸爸妈妈永远的支持使得我在学业上一帆风顺。他们虽然无法在知识上给我帮助，但他们的教诲是我前进的动力。爸爸妈妈对我的爱使得我牢记当前的幸福，并激励着我为了自己的理想而奋斗。