SPARQL 查询分析器的设计与实现

学　　院　软件学院

专　　业　软件工程

年　　级　2006 级

姓　　名　唐嘉泽

指导教师　张坤龙

2010 年 6 月 15 日

摘　要

近些年来，语义网的概念已经逐渐开始流行，对其所做的研究也日益增多。由于 RDF（资源描述框架）是语义网的核心概念，所以它成为了一个重要的研究内容。SPARQL 是由 W3C 提出的 RDF 数据的标准查询语言。

论文设计和实现了一个名为 SPACO 的 SPARQL 查询分析器。SPACO 是一个采用 Coco/R 语法分析生成器生成的运行在 Microsoft .NET Framework 环境下的可视化查询分析器，它能对输入的 SPARQL 查询语句进行相应的词法分析和语法分析、生成抽象语法树、并显示出具体的语法内容。

论文采用 W3C 给出的 SPARQL 语言语法测试用例对 SPACO 进行了测试。在测试过程中，针对不同的语法规则，对测试用例进行了分类，并分别进行了测试。

关键词：SPARQL；语法分析；抽象语法树；RDF；查询分析；

ABSTRACT

In recent years, Semantic Web has grown in popular. The size of the research is also gradually developing. As the core of the Semantic Web, RDF(Resource Description Framework) is always an important part of the study. SPARQL is a query language, recommended by W3C for querying RDF data.

This paper will give a SPARQL query parser's design and implementation, which is named SPACO. The parser has been created for Microsoft .NET Framework using the Coco/R complier generator. It is capable of scanning, parsing and generating the SPARQL queries into abstract syntax trees representing the queries.

The testing use cases for the parser are syntax tests for SPARQL language released by Data Access Working Group of W3C. For each category of grammars, test cases have been classified.

Key words：SPARQL；grammar parse；abstract syntax tree；RDF；query analysis；

目　　录

第一章　　绪论

1998 年互联网的创始人 Tim Berners-Lee 首次提出了语义网的概念。语义网旨在将传统网络中以应用程序为中心的数据形式，转换成可在多个数据源进行互联交互的数据形式。在语义网中，RDF（资源描述框架）是网络资源中数据的主要表示形式。

为了使网络资源数据能够得到更好的利用，需要提供一种对其进行查询的方法。RDF 数据访问工作小组针对 RDF 数据集提出了一种名为 SPARQL（SPARQL Protocol and RDF Query Language）的查询语言，并且该语言在 2008 年由 W3C 进行了标准规范。本文将要设计和实现一个 SPARQL 查询分析器。该查询分析器能够对读取的 SPARQL 查询语句进行词法分析和语法分析，生成一棵对应的抽象语法树。

1.1　RDF 概述

RDF 数据模型是一种与经典概念化模型（如实体-关系模型、类图模型等）相类似的数据表示形式。RDF 将各种数据资源，特别是网络资源，按照“主语-谓语-宾语”的三元组形式进行表示。其中主语代表资源，谓语代表主语与宾语之间的某种关系，即主语资源所具备的动作或属性等，例如如图 2-1 所示一个事实的表述为“计算机可以运行程序”，其中主语为“计算机”，谓语为“可以运行”，宾语为“程序”。RDF 的这种资源描述的机制是 W3C 所提出的语义网中的最主要内容之一。作为新一代的互联网平台，这种资源描述机制可以通过网络传输机器可自动识别的细心，并使软件对信息进行自动存储交换，这样用户就可以更方便有效的获取并处理信息。RDF 数据模式因其简介的风格，和其对不同模型的区分与概念抽象的能力，使得其在知识管理应用方面同样得到了广泛的使用。

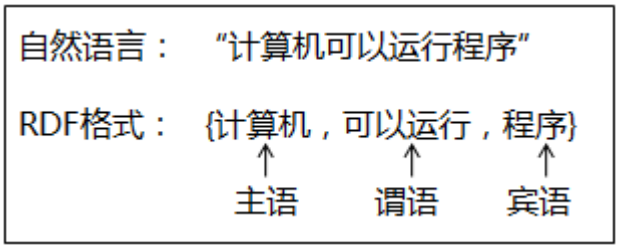

图 2-1　　RDF 数据表示格式

RDF 语句的集合从本质上讲可以表示为一个带有标签的有向多重图。与关系模型和其他的本体类模型相比，基于 RDF 的数据模型更加适合于进行各种知识的表达。

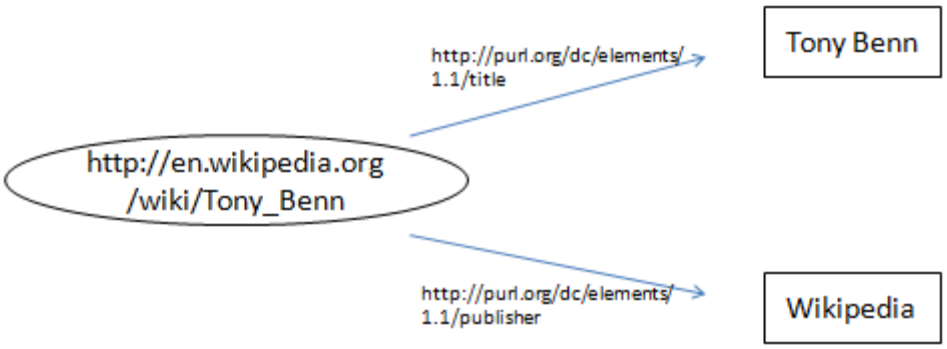

图 2-2　　RDF 示例图

图 2-2 给出了一个 RDF 语句集合的一个有向图，图中表示主语为 wikipedia 中以 Tony Benn 为标题的页面，通过两个页面的 URI 与宾语表示的标题与发行者的具体内容进行连接。在图中，主语和宾语作为节点，谓语作为二者之间某种关系的连接。

上述例子的 RDF 数据采用 XML 的格式将会得到如图 2-3 表示。XML 并非 RDF 数据的唯一表示方法，也可采用 Turtle 的方法进行表示，如图 2-4 所示。

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
      <rdf:Description
          rdf:about="http://en.wikipedia.org/wiki/Tony_Benn">
          <dc:title>Tony Benn</dc:title>
          <dc:publisher>Wikipedia</dc:publisher>
      </rdf:Description>
</rdf:RDF>
```

图 2-3　　XML 格式的 RDF 数据

```
@perfix a: <http://purl.org/dc/elements/1.1/>.

<http://en.wikipedia.org/wiki/Tony_Benn>
        a:title "Tony Benn";
        a:publisher "Wikipedia".
```

图 2-4　　Turtle 格式的 RDF 数据

1.2　SPARQL 概述

SPARQL（SPARQL 协议与 RDF 查询语言）是针对用户需求和 RDF 数据访问工作小组所制定的要求所设计的一种查询语言。SPARQL 查询语句中可以包含三元组模式、合取、吸取和选择模式等等。大部分的 SPARQL 语句中都包含一个类似于 RDF 三元组形式的元组模式集合，不同之处在于三元组中的某个属性

可能是以变量的形式出现。在查询过程中，可在 RDF 的数据结构图中查找与查询语句中的三元组所形成的查询图匹配的子图，用以获得查询语句中变量的具体值。

针对图 2-3 中所给出的 RDF 数据，在图 2-5 中给出了用以匹配“title”子图的 SPARQL 查询语句。查询的结果是用值“Tony Benn”替代了查询语句中的变量?title，返回的结果如表 2-1 中所示。

```
PREFIX a: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE
{
<http://en.wikipedia.org/wiki/Tony_Benn> a:title ?title
}
```

图 2-5　　SPARQL 查询语句实例

表 2-1　　查询结果

title
Tony Benn

针对不同的查询需要，SPARQL 查询语言规范中规定了一些模式和关键字方便用户的查询操作和对查询条件进行约束。“FILTER”方法可以通过对字符串采用正则表达式和对数值采用算数表达式进行查询过滤。RDF 三元组中的主语和宾语可以是以空白节点的形式出现。而在 SPARQL 查询语句中，查询结果中的空白节点并不要求与查询语句中的空白节点使用同一个标签。空白节点的作用即在查询过程中扮演不同变量的角色。所以用户不能将查询中的空白节点指定为数据中的某个特定的空白节点。

SPARQL 中给出“OPTIONAL”关键字，用来将 SPARQL 查询模式匹配多个不同的 RDF 查询子图。如果查询语句与多个 RDF 子图相匹配，则每个子图的结果都将会被返回。同样，也可以将多个 RDF 存储结构采用“OPTIONAL”的方法在查询中进行混合。

SPARQL 中还有一些与所熟知的 SQL 语言相类似的查询结果约束方法，如“Order”将结果进行排序；“Projection”在结果中返回特定的变量值；“Distinct”是结果中每一个返回值是唯一的。此外还包括一些其他的约束方法，如“Reduced”是在返回结果中消除某些重复的返回值，但并不是完全消除；“Offset”是在使用“Order by”的查询语句中决定结果从第几个返回值开始，同样“Limit”是在“Order by”的查询语句中决定结果返回值的数量。

SPARQL 中包含了四种查询模式：“SELECT”返回变量在查询模式中所匹配值的全集或子集；“CONSTRUCT”返回变量在查询模式中所对应的 RDF 查询

图；“ASK”返回查询模式是否匹配的布尔值；“DESCRIBE”返回匹配 RDF 查询图的具体资源内容。

1.3 论文结构

第二章将介绍 SPARQL 查询分析器的设计思路，主要包括语法分析生成器的选择，程序设计语言与编码环境的选择等等。第三章给出查询分析器的具体实现过程，包括针对 SPARQL 文法的语法分析生成器的使用，抽象语法树的结构实现，程序设计模式的选择，分析器的错误处理方法等等。第四章中通过使用 W3C 数据访问小组所给出的 SPARQL 测试数据，针对 SPARQL 的不同语法种类进行测试，并给出相应的测试结果。第五章将对本文所提出的 SPARQL 查询分析器的设计与实现过程进行归纳总结，并对未来的工作提出设想。

第二章　SPACO 的设计

2.1　语法分析器生成工具的选择

设计与实现查询分析器的一个关键环节就是选择一个合适的语法分析生成器。语法分析生成器根据一个使用巴科斯范式描述的形式文法来生成高级语言代码，即词法与语法分析器。将要实现的系统是采用 C#开发语言，所以选择的生成器必须能够生成 C#语言的程序文件。并且根据 W3C 的 SPARQL 规范中所提到的，“SPARQL 采用大写字母作为终结符名称时是 LL（1）文法”[1]，分别对几个比较适合的生成工具进行分析与比较。

2.1.1　ANTLR

ANTLR（Another Tool for Language Recognition）是一个开源的语法分析器生成工具。采用递归下降 LL（*）文法进行语法分析，支持 C，Java，C#，Python 和 Objective-C 语言。形式文法采用扩展巴科斯范式 EBNF。

2.1.2　MPLex 和 MPPG

MPLex（Managed Package Lex）和 MPPG（Managed Package Parser Generator）是由微软在 Visual Studio SDK 中集成的词法与语法分析器的生成工具。由 MPLex 生成词法扫描器，MPPG 生成语法分析器。这种生成工具专门支持 C#语言，采用自底向上的 LALR 文法进行语法分析。

MPLex 和 MPPG 所使用的文法格式与经典的 Lex 和 yacc 生成工具相类似，形式文法采用普通的巴科斯范式 BNF。

2.1.3　Coco/R

Coco/R（Compiler compiler/Recursive descent）也是一个开源的语法分析器生成工具。支持 C#，Java 等多种语言，且是在 C#语言方面较为流行的一种生成工具。Coco/R 采用 LL（*）文法进行语法分析，形式文法采用基于属性的扩展巴科斯范式。

2.1.4　生成工具的选择

表 3-1　3 种生成工具的比较

	ANTLR	MPLex&MPPG	Coco/R
形式文法	EBNF	BNF	基于属性的 EBNF
语法分析类型	LL(*)	LALR	LL(*)
抽象语法树生成	自动生成	手动编写	手动编写
开发环境的支持	无	VS Managed Babel	VS Plug-in

表 3-1 中给出了之前提到的 3 种生成工具的比较结果。根据 W3C 提供的 SPARQL 查询语言规范，SPARQL 的规范文法采用的是扩展巴科斯范式，并且在以大写字母作为终结符的时候采用 LL（1）文法，所以将工具的选择范围缩小到 ANTLR 和 Coco/R。由于设计的查询分析器采用 C#语言进行开发，并且在开发环境 Visual Studio 中有 Coco/R 的插件进行配合，方便开发，因此最终选择了 Coco/R 作为该分析器的生成工具。

2.2 开发语言与环境的选择

正如前面所提到的，本文所要设计的分析器要对 SPARQL 查询语言进行词法分析、语法分析，并生成抽象语法树（AST）。如果程序采用经典的 Lex 和 yacc 生成工具，则抽象语法树的构建将无法采用面向对象的设计模式，而是使用 C 语言中的结构体。通过这种方式编写的程序将很难适应更复杂、更庞大的数据处理。

选择 C#，一方面它可以兼容 C 语言中的结构体，另一方面它可以采用面向对象的形式，用类作为抽象语法树的数据结构。C#中的结构体只能采用值的方式进行表示，结构体只允许执行接口，而无法继承类和其他结构体。因此 C#中的结构体也并不适合用于实现抽象语法树。一个抽象语法树的生成本质上就是树节点之间的相互引用，所以面向对象形式中的类是实现抽象语法树节点表示的唯一形式。类于类之间的继承和与接口之间的连接可以很好的实现抽象语法树中节点之间相互关联的特性。

为了更好的使用 C#开发语言，编程选用了最为适用的 Windows 操作平台下的 Visual Studio 集成开发环境。其中 Visual Studio 中所集成的测试功能也为后续的测试工作提供了很好的条件，具体测试内容将在第五章进行详细介绍。

2.3 SPACO 的结构

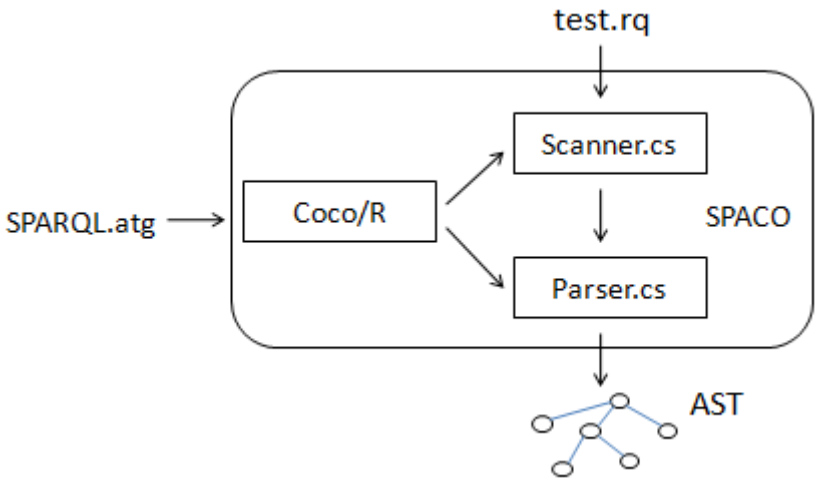

图 3-1　　SPACO 的结构

图 3-1 即为名为 SPACO（SParql query Analyzer with COco/r）查询分析器的设计流程。要先根据 Coco/R 生成工具的形式文法编写 SPARQL 语言的文法文件，通过 Coco/R 的解析生成 SPARQL 查询语言的词法扫描器 Scanner.cs 和语法分析器 Parser.cs；针对读取的 SPARQL 查询语句，对其进行词法扫描和语法解析，之后根据语法规范，构建该查询语句的抽象语法树。

2.3.1 SPARQL 文法解析阶段

在 SPARQL 文法文件的编写过程中，要参照 W3C 给出的 SPARQL 文法规则，将关键字、终结符和非终结符等各条文法，编写到对应的 Coco/R 文法文件中，同时为了符合 Coco/R 中的扩展巴科斯范式，需要对具体的文法规则进行符号的翻译。并且为了使所生成的词法分析器和语法分析器更加有效正确的运行，在文法文件中针对不同的文法给出语义表示的 C#目标代码，以辅助分析器的生成。

在 Visual Studio 所支持的 Coco/R 插件中，编译环境会自动生成文法文件的模板，根据模板中的内容进行编写，很大程度的减少了文法文件编写的困难。并且插件中会自动对所编写的文件进行编译，对错误和警告给出相应的提示。

在 Coco/R 生成分析器的过程中，还需要两个框架文件进行支持，也可以针对自己系统的需要对框架文件进行修改。

2.3.2 查询语句分析阶段

在生成了词法语法分析文件之后，就要编写调用程序，来完成下一步的操作，即读取 SPARQL 查询语句，生成对应的抽象语法树。为了最终所生成的抽象语法树，在生成语法分析器的时候，就已经针对不同种类的节点给出了相应的建立树节点的代码。

在调用程序中，将创建树节点的通用接口，通过对接口的实现和对节点类的继承，创建不同种类的语法节点，并通过对象的实例化创建每个查询语言所解析的具体节点内容。在这个阶段，还采用了访问者模式的设计思路，针对不同种类的节点，在不改变类中的属性和方法的前提下，根据不同种类的节点，在类的对象中对每种节点执行不同的操作。具体的实现过程将在第三章中进行详细介绍。

2.4 SPACO 的测试

设计一个可视化的测试工具，即通过读取具体的 SPARQL 查询语句文件，进行词法扫描后，在工具中返回对语句各个分词的结果；进行语法分析后，工具中将给出具体的树形结构图，树中的每个节点的具体内容也将对应的显示出来。同时对于解析过程中的错误，也可以通过工具将错误的具体内容进行返回。

基于 Visual Studio 中的强大支持，测试工具采用 Windows Forms 控件作为平台，因为 W3C 的 SPARQL 文法规范中没有明确给出抽象语法树的具体形式，所以在测试工具中对抽象语法树的可视化过程采用 TreeView 对象进行实现。

第三章　SPACO 的实现

本章将对 SPACO 分析器的实现过程进行详细的介绍。其中主要分为以下几部分内容进行介绍：开发环境配置；SPARQL 规范针对 Coco/R 生成器的形式文法创建；生成抽象语法树的设计模式；词法、语法分析过程中的错误收集处理；SPACO 测试客户端的设计与实现等等。

3.1　开发环境的配置

SPACO 查询分析器所使用的开发环境为 Visual Studio 2008，辅助的语法分析器生成工具为 Coco/R，两种工具可在以下的网站通过下载获得：

Microsoft Visual Studio 2008，在微软的 MSDN 学术联盟中提供 90 天的使用版本下载：http://msdn.microsoft.com/en-us/vstudio/products/aa700831.aspx；在微软学生中心也提供专门为高校学生准备的 VS2008 速成版本的下载：http://www.msuniversity.edu.cn/static/vse2008/default.html。

Coco/R 是由奥地利约翰开普勒林茨大学的系统软件研究所设计开发的。Coco/R 针对各个语言版本的源码、工具、说明书和测试数据都可在该大学的官方网站下获得下载：http://www.ssw.uni-linz.ac.at/Coco。需要使用的工具主要是包括 Coco.exe 的可执行文件和 Scanner.frame、Parser.frame 两个生成文件的框架。

在 Visual Studio 解决方案中，首先要建立其为分析器自动生成的工程文件 Project File，该工程文件与 Unix 系统中开发使用的 makefile 相类似，都是用来描述工程中各种建立操作的具体内容。

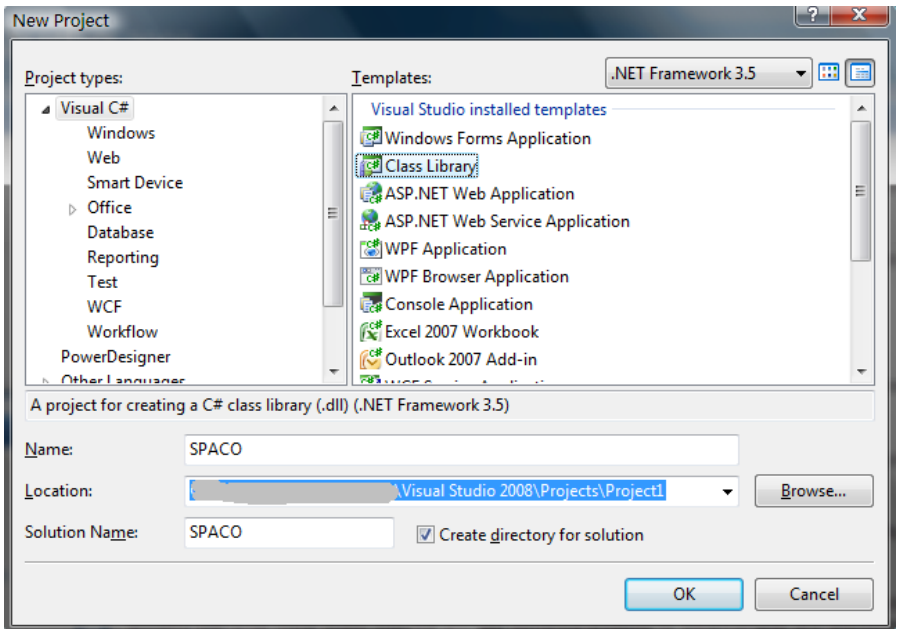

图 4-1　建立 Visual Studio 工程文件

如图 4-1 所示，首先要在 Visual Studio 中创建 SPACO 的工程文件，即点击

File→New→Project…，然后选择 Class Library 选项，工程文件命名为 SPACO。

删除掉自动生成的 Class1.cs 文件。之后要在工程中加入将要使用的文件。在生成词法和语法分析器的过程中，将要使用到 Coco/R 组件中的 coco.exe 可执行文件和 Scanner.frame、Parser.frame 框架文件，coco.exe 可在系统环境变量中进行设置。框架文件可通过在工程中选择 Add→Existing Item…进行添加。之后要创建一个名为 SPARQL.atg 的形式文法文件，该文件用来让 Coco/R 生成工具解析 SPARQL 文法，从而创建词法和语法分析器。

词法与语法分析文件的生成过程可通过使用 System.Diaglostics.Process 类库调用 coco.exe 文件。如图 4-2 中的示例：

```
System.Diaglostics.Process p = new System.Diaglostics.Process();
p.StartInfo.FileName="coco.exe";
p.StartInfo.Arguments="%ProjectFolder%\SPACO\SPARQL.atg -namespace SPACO";
p.Start();
if(p.HasExisted)
  p.kill();
```

图 4-2　调用 coco.exe 程序，解析 SPARQL.atg 文件

词法扫描器还需要调用一个 IErrorHandler 接口用来对输入文件中的错误内容进行交互处理。因此需要在 SPACO 工程文件中创建一个接口文件，命名为 IErrorHandler.cs，并且设置为 public 访问类型。在解决方案中工程的布局如图 4-3 所示。

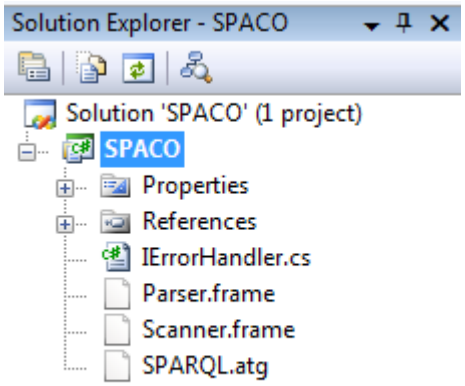

图 4-3　初始配置后的解决方案示例

3.2　抽象语法树

Coco/R 中并不支持自动生成的抽象语法树。为了在分析器中能够生成 SPARQL 语句的抽象语法树，需要在文法文件的语义操作部分创建并构造语法树的节点对象。节点对象在节点类中进行了创建，节点类遵循抽象语法树的类层次模型。每种节点类的具体实现都包含在 SPACO.Ast 命名空间中。

3.2.1 抽象语法树类层次模型的设计

树的类层次模型的根部是一个简单的接口 INode，如图 4-4 所示。该接口对树的创建提供了最基本的支持，其中通过 Parent 和 Children 两个属性从树节点的两个不同方向进行树的构造。Accept 方法作为 Visitor 接口的一部分，将在 4-4 节中进行详细介绍。

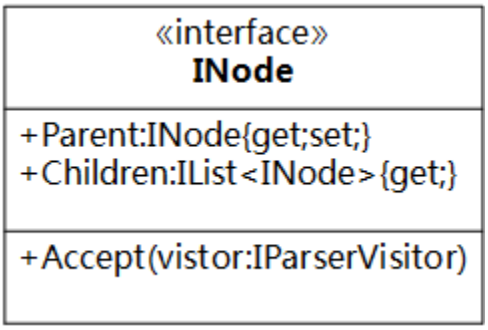

图 4-4　　INode 接口图

在 INode 接口下面就是节点的抽象类 NodeBase，如图 4-5 所示。这个类的作用是给出了对接口 INode 的基本实现。其中每个属性和方法都采用了 visual 的形式，目的是在必要时可使子类对其属性和方法进行重写。该抽象类中虽没有抽象成员，但其被作为抽象类处理可避免被实例化的可能，而且为后续工作中可能添加的抽象成员提供了便利。

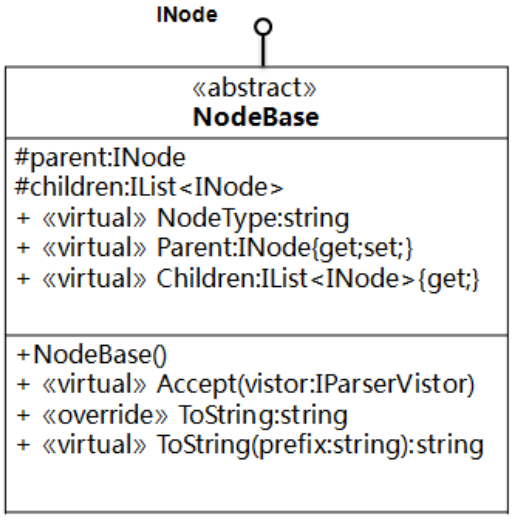

图 4-5　　NodeBase 抽象类图

Parent 和 Children 属性通过 parent 和 children 变量与域进行取值与赋值的操作。只读属性 NodeType 作为每个类的唯一标识，是访问者模式中的一部分。基本的实现方法是使用反射机制检索到对应节点类的名字。如果需要，这个属性应该在子类中进行重写。

NodeBase 给出了两个构造函数。默认无参数的构造函数是使 children 域的初始化为一个空的集合；第二个构造函数是使其初始化为带有参数 NodeBase 对象的集合。

NodeBase 中 Children 属性的返回类型是 INode 接口的一个序列，自定义的 NodeCollection 类实现了这个序列，并定义了添加、删除、索引等方法。

每个节点类都包含在 SPACO.Ast.Nodes 命名空间中。这些类都是从 NodeBase 派生出来，因此也都实现了 INode 接口。

3.3 文法描述

在前文中曾经提到过，SPARQL 的规范文法采用的是扩展巴科斯范式，并且在以大写字母作为终结符的时候采用 LL（1）文法。传统的巴科斯范式与扩展巴科斯范式的主要区别是对？（0 或 1）、*（0 或多个）和+（1 或多个）符号的支持。所采用的生成工具 Coco/R 可以很好的支持扩展巴科斯范式。在 Coco/R 中主要使用的符号如表 4-1 所示。这就很好的解决了扩展巴科斯范式的问题。

表 4-1　Coco/R 中采用的基本符号

符号	含义	例子
=	分隔文法规则的两端	A = a b c.
.	文法规则的中止符号	A = a b c.
|	分离替换	a b | c | d e，表示 a b 或 c 或 d e
( )	分组替换	(a | b) c，表示 a c 或 b c
[ ]	选择	[a] b，表示 a b 或 b
{}	迭代（0 或多次）	{a} b，表示 b 或 a b 或 a a b 或…

3.3.1 SPARQL 文法概述

在 Coco/R 中，生成语言的文法规范主要分为四个部分：字符、标识符、编译标注和文法规范产生式。字符部分是对在 SPARQL 语言中可能出现的字符进行形式化说明，如图 4-6 所示。标识符即使对 SPARQL 文法中的关键字与终结符进行形式化说明，如图 4-7 所示。

在文法规范产生式部分，就是对 SPARQL 语法规范的非终结符文法进行转换，其中可添加相应的语义操作代码，辅助语法生成器的生成工作。图 4-8 给出了针对语法规范中的第 13 条 Where 语句的文法规范产生式。

```
CHARACTERS
letter='A'..'Z' + 'a'..'z'.
digit = "0123456789".
cr='\r'.
lf='\n'.
tab='\t'.
stringCh=ANY- '"' - '\\' - cr - lf.
charCh=ANY - '\'' - '\\' - cr - lf.
printable = '\u0020' .. '\u007e'.
hex = "0123456789abcdef".
```

图 4-6　　字符形式化声明

```
TOKENS
INTEGER = [0-9]+
DECIMAL = [0-9]+ '.' [0-9]* | '.' [0-9]+
DOUBLE = [0-9]+ '.' [0-9]* EXPONENT | '.' ([0-9])+ EXPONENT | ([0-9])+ EXPONENT
INTEGER_POSITIVE = '+' INTEGER
DECIMAL_POSITIVE= '+' DECIMAL
DOUBLE_POSITIVE = '+' DOUBLE
INTEGER_NEGATIVE = '-' INTEGER
DECIMAL_NEGATIVE = '-' DECIMAL
DOUBLE_NEGATIVE = '-' DOUBLE
EXPONENT = [eE] [+-]? [0-9]+
```

图 4-7　　标识符形式化声明

```
/* [13] WhereClause ::= 'WHERE'? GroupGraphPattern*/

WhereClause =
 'WHERE' GroupGraphPattern (. new WhereClauseNode(la.next) .)
| GroupGraphPattern (. new WhereClauseNode(la) .)
```

图 4-8　　SPARQL 形式文法产生式示例

3.3.2　词法扫描器说明

用于生成词法扫描器的标识符文法采用正则表达式进行表示。每个标识符都在分析器中得到返回，并且分别定义一小段代码用来返回分析器可识别的相关类型的标识符集合。此外，用于创建其他标识符的辅助标识符也可被定义，这样就避免了正则表达式的重复出现。

在标识符的文法定义过程中，声明的顺序是一个必须要考虑到的问题。在标识符匹配的过程中，词法扫描器会尝试匹配最长可能的标识符。如果多个标识符

间存在约束关系，那么最先定义的那个标识符将被返回到分析器中。而对于 SPARQL 语法，标识符的声明顺序可以忽略。所有的标识符共享一个通用的不同长度前缀，因此约束关系是不会产生的。

3.3.3 语法分析器说明

文法规则用于生成语法分析器。每个文法规则可以使用一个或多个形式文法公式进行定义，这些形式文法可以返回一个标识符的值或者是与其相关联的抽象语法树节点。当输入的字符串不足以确定的时候，文法内容也可以设置为空。

根据 W3C 的 SPARQL 文法规范，将其转化为 Coco/R 可识别的形式文法，并且结合分析器中生成抽象语法树的语义要求，针对每条文法规则给出相应的程序代码。

3.4 访问者模式

访问者模式，顾名思义就是在可以不修改已有程序结构的前提下，通过添加额外的访问者来完成对已有代码功能的改进。访问者模式的本质就是要将算法与数据结构相分离，在不改变类中操作的前提下，通过对象结构定义新的操作。

在的分析器系统中，访问者模式在抽象语法树的建立过程中起到了很大的作用。在 SPACO 分析器所生成的抽象语法树中的每个节点对象都有一个名为 Accept 的方法，该方法实现了 IParserVisitor 接口，如图 4-9 所示。

图 4-9　　IParserVisitor 接口

接口中只有一个成员函数，该函数以一个标识节点类型的字符串作为输入，返回一个代理动作集 Action<INode>。该动作集可作为一个新创建方法的指针，其指向 INode 变量所接受到的各个方法。抽象语法树中的节点所接受的访问者将会访问这个成员函数，讲该节点的 NodeType 传递给该函数作为输入，并调用代理获得返回值。具体的 Accept 函数的代码片段如图 4-10 中所示。访问者是按照深度优先的顺序遍历树的节点，也就是说在访问某个内部节点的时候已经对其子节点遍历过了。

```
public virtual void Accept(IParserVisitor visitor)
{
    foreach (INode child in children)
    {
        if (child != null)
            child.Accept(visitor);
    }
    visitor[NodeType](this);
}
```

图 4-10　NodeBase 访问者模式的基本实现

3.5　错误处理

SPACO 查询分析器中的错误处理机制是针对输入的 SPARQL 查询与相关的错误条件进行判断，并报告出所出现的错误内容。词法扫描器的基本功能是识别出查询输入流中的标识符并将其传递给语法分析器，之后使用定义的文法规则进行语法分析。如果出现了没有定义过的标识符，或者是缺少或错误的标识符，则会出现一个像匹配的错误条件，由错误处理机制进行处理。

3.5.1　传统错误处理方法

在传统的词法语法分析器生成工具中，如 Lex 和 yacc，一般都是在词法扫描器确定输入流中存在明显的错误后立刻给出错误报告，或是在将识别出的标识符传递到语法分析器的过程中由语法分析器对错误的标识符进行报告。但是由于在词法扫描器中判断出的错误只局限于标识符本身的错误，而在语法分析器中则不仅能判断出标识符所存在的问题，而且还可以给出分析器所预期的标识符是什么，这样有助于错误的修改。

通过将词法扫描器收集到的错误信息，包括错误标识符的内容和所出现的具体位置，与语法分析器所识别出的错误标识符信息相结合，就会得到一个完整有效的错误处理信息，这也是在 SPACO 查询分析器中所采用的错误处理机制。

3.5.2　SPACO 中的错误处理

SPACO 分析器所采用的 Coco/R 生成工具中，将错误处理作为生成的语法分析器的一部分，Coco/R 中定义的错误处理类 Errors 框架如图 4-11 所示。其中变量 count 表示 SynErr 与 SemErr 中出现错误的数量总和；errorStream 是处理错误报告的输出形式；errMsgFormat 是对错误报告的具体格式进行设置，报告的内容可包括错误出现的具体位置和错误的主要内容；语法错误使用 SynErr 方法处理，语义的错误通过 Parser.SemErr 传递到 Errors 类中的 Errors.SemErr 方法进行处理；警告则使用 Warning 方法进行处理。SynErr 和 SemErr 方法的参数作为错误出现

的行列位置传递给 errorStream 错误格式变量，其中{0}和{1}分别表示行列数，{2}表示错误的具体内容信息。

```
class Errors {
      public int    count = 0;
      public string errorStream  = Console.Out;
      public string errMsgFormat = "-- line {0} col {1}: {2}";
      public void   SynErr(int line, int col, int n);
      public void   SemErr(int line, int col, string msg);
      public void   SemErr(string msg);
      public void   Warning(string msg);
}
```

图 4-11　　Errors 类框架

在生成的词法分析器和语法分析器中，标识符的数据结构采用结构体，其中定义了标识符的类型、所在行、所在列和标识符的值。在 Parser 类中通过 SynErr 方法将错误的标识符的行、列及错误内容作为参数传递到 Errors.SynErr 方法，对错误进行处理。

3.6　SPACO 的执行流程

图 4-12 给出了 SPACO 查询分析器的 UML 顺序图。图中显示了 SPACO 在词法和语法分析阶段的整个执行流程。

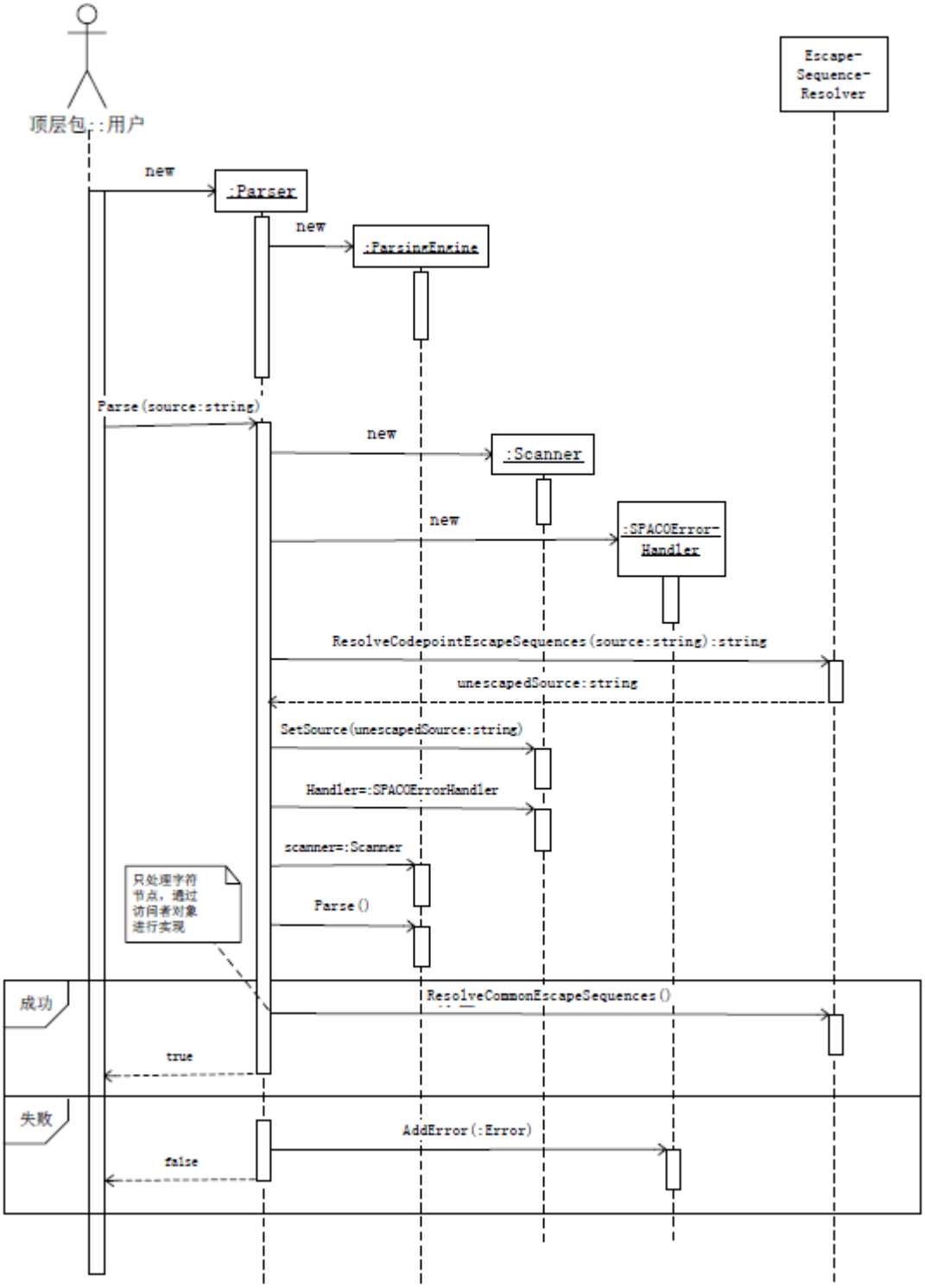

图 4-12　　SPACO 词法、语法分析的 UML 顺序图

第四章　SPACO 的测试

4.1　SPACO Test Client 的创建

SPACO 测试客户端是通过调用 SPACO 分析器的类库，对读取的 SPARQL 查询语句进行解析，之后给出所生成抽象语法树的可视化表示。该客户端采用 Visual Studio 中的 Windows Forms 组件作为设计平台，由于 VS 对 Windows Forms 的强大支持，使得客户端可以很容易的建立图形化的用户界面与接口。在 SPACO 的解决方案中添加一个新的 Windows Forms 工程,其中要引用之前创建的 SPACO 类库，命名为 SPACOTestClient，如图 5-1 所示。

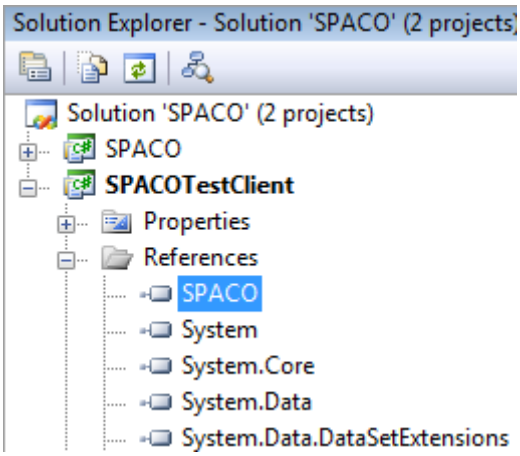

图 5-1　测试客户端的引用内容

图形用户界面的基本设计如图 5-2 所示，点击“Browse…”按钮读取写有 SPARQL 查询语句的测试文件，点击“Scan”按钮对读入的文件进行词法分析，弹出如图 5-3 的结果窗口，返回分析结果。点击“Parse”按钮则对读入的查询文件进行语法分析，分析得到的抽象语法树结果显示在下方左边的窗口内，并以 TreeView 的树状结构显示。当选中树中某个节点的时候，该节点的具体内容将对应的显示在右侧的窗口中。读取、解析文件过程中的错误都将显示在下方的表格中。

由解析生成的抽象语法树转变成具体可视化的树节点的过程在之前 4.4 中所介绍的访问者模式中已经得到了解决。使用这种模式，转变的过程十分简便。TreeBuilderVisitor 类首先建立 INodes 对象与 TreeNode 对象的索引。因为访问者模式是采用深度优先的方法对树进行遍历，所以当访问者访问到某个节点的时候，就说明它已经对其子节点都进行了访问。

访问者无法区分不同种类的节点，他们每次返回的都是相同的代理，该代理执行的方法如图 5-4 中的代码所示，通过方法中的 NodeType 属性获得执行该方法的类的名称，从而得到节点的种类。

将抽象语法树中的节点添加到 TreeNode 节点中的“Tag”属性中，之后该节点的具体内容也将添加到对应的对象信息中，这样就可以得到对应树节点的具体内容，并在客户端的右侧把相应的结果显示出来

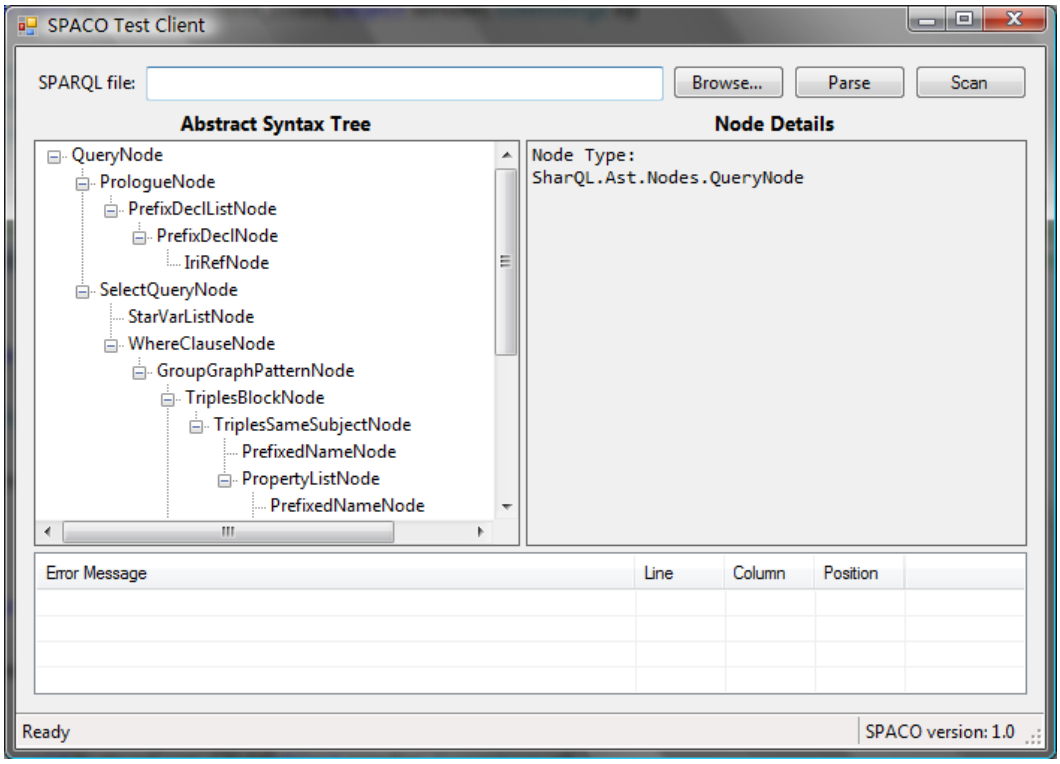

图 5-2　　测试客户端的图形用户界面

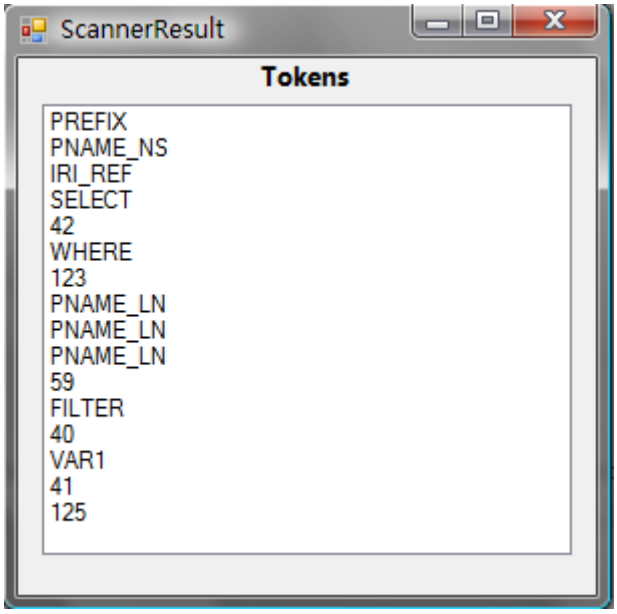

图 5-3　　词法分析结果示例

```
private void constructTreeNode(INode node)
{
        NodeBase nb = node as NodeBase;
        if (nb != null)
        {
            // TreeNodes maps INode objects to TreeNode objects
            TreeNodes[node] = new TreeNode(nb.NodeType.Split('.').Last());
            TreeNodes[node].Tag = node;
            foreach (INode child in nb.Children)
            {
                if (child != null && TreeNodes.ContainsKey(child))
                {
                        TreeNodes[node].Nodes.Add(TreeNodes[child]);
                }
            }
        }
}
```

图 5-4　　TreeBuilderVisitor 代码片段

4.2　Visual Studio 中的自动测试功能

Visual Studio 中整合对自动测试功能的支持。在解决方案中添加一个测试工程，并可编写多种语言的测试程序，由系统自动测试，并对测试结果返回报告。

在测试工程中进行单元测试，首先要创建一个如图 5-5 的公有类，并添加 TestClass 标签。当 Visual Studio 决定运行测试工程中的测试时，会通过反射机制决定哪个类是测试类。

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
(...)
[TestClass]
public class MyUnitTests
{
        (...)
}
```

图 5-5　　测试类的创建

在测试类中可以定义任意的测试方法，这些方法可以对测试对象的状态、结果进行反馈。与测试类同理的是，测试方法也需要具有 TestMethod 的属性标签。其中可使用一些静态的方法，如 Assert 类，对结果进行返回。图 5-6 给出了一个具体的实例，其中创建了一个测试对象，并对其状态进行了返回。该方法针对两个条件进行判断，给出相应的提示结果。

```
[TestClass]
public void MyUnitTest1()
{
    MyClass myObj = new MyClass();
    myObj.DoSomething(1, 2, 3);
    Assert.IsTrue(myObj.SomeProperty == 5);
    Assert.IsFalse(myObj.AnotherProperty > 0);
}
```

图 5-6　测试方法的创建实例

编写好测试工程后可通过在 Visual Studio 中点击 Test→Run→All Tests in Solution 运行测试。

4.3　W3C 的 SPARQL 测试组件

W3C 的数据访问工作小组给出了一个测试用例的集合，其中包括 SPARQL 语言的语法测试。在全部的测试用例中，包含 199 个 SPARQL 的查询语句用例，在这些用例中一部分是可解析成功的，另一部分是有错误而解析失败的用例。虽然这些用例不能全面的覆盖到 SPARQL 语言的每个细节，但是它为 SPARQL 语法的测试提供了一个很好的基础。

这些测试用例并不是简单的 Select/Where 语句，而是如图 5-7 所示的样子。其中也包括如图 5-8 所示的无效查询语句，这些用例涵盖了 SPARQL 语法中的大部分规则。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
CONSTRUCT { []   rdf:subject ?s ;
                 rdf:predicate ?p ;
                 rdf:object ?o }
WHERE {?s ?p ?o}
```

图 5-7　W3C syntax-form-construct03 测试用例

```
SELECT * WHERE {[] . }
```

图 5-8　W3C syn-bad-bnode-dot 测试用例

因为 W3C 提供的测试组件没有明确的给出抽象语法树的结构，所以它无法对所产生的抽象语法树给出确定的评价。并且也没有单独的给出词法分析的评价标准。所以使用这些测试用例，然后人工的对测试用例与测试结果进行比较与评判。

4.3.1 忽略的语义测试用例

在 W3C 给出的测试组件中，其中一部分是针对语义的测试用例，在表 5-1 中给出。这些用例的正确测试结果应该是解析失败，但是在分析器测试过程中的结果是解析成功。造成这种现象的原因是原本这些用例中的一些空白节点是通过基本图模式进行解析的，但是分析器不支持这样的语义上的错误问题，所以忽略了这个错误。所以在最后应用的 199 个测试用例中删除了 11 个这样的语义测试用例，而最终采用了 188 个测试用例进行测试。

表 5-1　　忽略的语义测试用例

测试文件	解析失败的原因
syntax-sparql3\syn-blabel-cross-graph-bad.rq	空白节点 who 通过基本图模式进行了重用
syntax-sparql3\syn-blabel-cross-optional-bad.rq	
syntax-sparql3\syn-blabel-cross-union-bad.rq	
syntax-sparql4\syn-bad-34.rq	空白节点 a 通过基本图模式进行了重用
syntax-sparql4\syn-bad-35.rq	
syntax-sparql4\syn-bad-36.rq	
syntax-sparql4\syn-bad-37.rq	
syntax-sparql4\syn-bad-38.rq	
syntax-sparql4\syn-bad-graph-breaks-BGP.rq	
syntax-sparql4\syn-bad-opt-breaks-BGP.rq	
syntax-sparql4\syn-bad-union-breaks-BGP.rq	

4.4 自定义测试

除了使用 W3C 提供的测试组件之外，还创建了一些自定义的测试用例用来测试分析器的其他方面。自定义测试用例的目的是可以独立的对分析器中的各个部分进行测试，获得更好的测试结果。

4.4.1 词法扫描器的测试

词法扫描器在分析器中起到的作用是对输入的 SPARQL 查询语句进行标识符的分析与识别。测试词法扫描器的一个最普通的方法就是针对某一查询语句词法扫描器返回的标识符的内容和顺序是否正确。针对 W3C 的测试组件中符合 SPARQL 语法各个方面的测试用例，自定义测试都可以执行，并返回标识符的结果。每个测试返回的标识符类型与顺序都为正确则说明测试成功。

图 5-9 给出了一个自定义词法扫描器的测试方法，以图 5-8 中的内容作为测试用例的话，测试返回的标识符结果应为“SELECT，*，WHERE，{，}”，并且顺序也是如此。

```
[TestMethod]
public void Scanner_1_basic_01()
{
        Assert.IsTrue(assertScannerOutput(
                @"..\..\..\Tests\syntax-
                sparql1\syntax-basic-01.rq",
                new Tokens[]
                {
                    Tokens.SELECT,
                    (Tokens)((int)'*'),
                    Tokens.WHERE,
                    (Tokens)((int)'{'),
                    (Tokens)((int)'}')
                    })
                );
}
```

图 5-9　　自定义的扫描器测试方法

4.4.2　转义测试

转义序列的处理在语法分析的前后都有相应的操作。一系列的自定义测试对这些操作都分别进行了测试。转义序列的处理方法和访问者也都分别进行了测试。图 5-10 给出了一个针对转义序列处理器的具体测试。

```
string escapedString = @"\u0061";
string expected = "a";
string actual;
actual = EscapeSequenceResolver.ResolveCodepointEscapeSequences(
            escapedString);
Assert.AreEqual(expected, actual);
```

图 5-10　　转义序列处理器的测试

第五章　结论

5.1　结论

本文所给出的 SPARQL 查询分析器的设计与实现过程，主要是将 Coco/R 语法分析器生成工具整合到 SPACO 分析器中，从而大大简化了语法分析器的生成过程，并且很好的支持了 SPARQL 查询语句的词法、语法分析与抽象语法树的生成。

将 SPARQL 规范中给出的文法转换成 Coco/R 形式文法的过程，是本次查询分析器实现过程中耗时最长，且最为复杂的一部分工作。而且转换的过程中所出现的程序 Bug 也是整个任务开发中最多的一部分。

为了是查询分析器的分析过程更加符合 SPARQL 的语法规范，就需要不断的消除程序中存在的 Bug，而 W3C 所提供的 SPARQL 测试组件则为此提供了很好的帮助。其中将近 200 个的测试数据应用到了 SPACO 分析器中，针对不同种类的语法进行测试。而除此之外，还有一小部分测试数据是用来针对语义的内容进行测试。

通过前文所提到的设计与实现的过程，最终得到了这个名为 SPACO 的 SPARQL 查询分析器。它运行与.NET Framework 环境平台上，并生成 SPARQL 查询语句的抽象语法树。W3C 所提供的全部测试数据都已通过，符合 SPARQL 语法规范中的指示级别。

5.2　展望

SPACO 作为一个 SPARQL 的查询分析器，可以与 RDF 数据库进行整合，作为 RDF 数据引擎的工具，下一步即可针对分析结果与 RDF 数据模式生成查询计划，SPACO 生成的抽象语法树可作为查询计划的基础；针对 RDF 数据存储与索引模式、语义要求等等方面，可对查询计划进行优化，从而设计一个较为强大的 RDF 数据引擎。

参考文献

[1]W3C：SPARQL Query Language for RDF[S]. http://www.w3.org/TR/rdf-sparql-query/.

[2]W3C：Resource Description Framework (RDF) [S]. http://www.w3.org/RDF/.

[3]Wikipedia，SPARQL[S]. http://en.wikipedia.org/wiki/SPARQL.

[4]Thomas Neumann，Gerhard Weikum，RDF-3X：a RISC-style engine for RDF[J]，Proceedings of the VLDB Endowment， v1 n.1, August 2008.

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

[6]Toby Segaran，Colin Evans，Jamie Taylor．Programming the Semantic Web[M]．USA：O'Reilly Media．2009.

[7]Grigoris Antoniou，Frank van Harmelen．A Semantic Web Primer , 2nd edition [M]．USA：MIT Press．2008.

[8]Markus Stocker，Andy Seaborne，Abraham Bernstein，Christoph Kiefer，Dave Reynolds．SPARQL basic graph pattern optimization using selectivity estimation[J]．New York，NY，USA．2008.

[9]陈彦．SPARQL 查询引擎设计[J]．电脑知识与技术（学术交流），2007，2-10.

[10]Krys J.Kochut，Maciej Janik．SPARQLeR: Extended Sparql for Semantic Association Discovery[M]．Springer Berlin / Heidelberg．2007.

[11]Alfred V. Aho，Monica S.Lam，Ravi Sethi，Jeffrey D. Ullman．编译原理[M]．北京：机械工业出版社．2008.

[12]John R.Levine，Tony Mason，Doug Brown．Lex 与 yacc，第二版[M]．北京：机械工业出版社的．2003.

[13]Hanspeter Mössenböck．The Compiler Generator Coco/R，User Manual[Z]．Johannes Kepler University Linz．2006.

[14]Ole Petter Bang, Tormod Fjeldskår．Developing a SPARQL parser for .NET [D]．TDT4510 Data and Information Management, Specialization Project．2008.

[15]W3C．DAWG Testcases[S]．http://www.w3.org/2001/sw/DataAccess/tests/r2.

[16]Compiler compiler/Recursive descent (Coco/R). http://www.ssw.uni-linz.ac.at/Research/Projects/Coco/，Oct. 2009.

致　　谢

能够顺利的完成这篇论文，我们感谢很多人。

首先感谢我的导师——张坤龙老师。张老师一直是我很钦佩的人，也是我在大学四年的学习和生活中最喜爱的一位老师。张老师认真负责的态度，严谨的研究风格，使我对张老师倍感钦佩。在毕业设计的整个过程中，张老师为我安排周密的学习与工作计划，在每个阶段中都定期对工作成果进行检查与批改。在后期的毕业论文编写过程中，张老师也是一次次的对我的论文进行的认真的修改，每一个章节，每一个段落，甚至是每一个子句都会认真的推敲，对格式的修改态度更是一丝不苟。同时在整个的过程中，张老师深厚的理论功底和高超的学术水平更加使我钦佩不已。这次毕业设计任务的完成，很大的功劳要归功于张坤龙老师。感谢张老师对我的悉心教导。

其次还要感谢孙博师兄，在任务开始的学习阶段，孙博师兄耐心细致的对我所学的知识进行指导，同时他也是我学习中很好的伙伴，我们的共同学习与努力才换来的今天的结果。

还有一个很重要的人，就是要感谢徐倩倩同学，在整个毕业设计的过程，是她在生活上给予了我无微不至的照顾，她的悉心陪伴与鼓励，成为了我精神上的动力，才能让我顺利的完成了这次毕业设计任务。

还要感谢我的父母，在这3个月的时间里因为毕设的事情，很少能够回家看望他们，但是他们只是默默的支持着我，没有任何的抱怨。

同时还要感谢同在一个宿舍生活四年的郭旭波、郭岭、刘树雷、袁文强和金晓军同学。我们是学习上伙伴，也是生活中的知己。在毕业设计的过程中，大家互相支持，共同讨论，互相帮助。非常感谢他们四年的陪伴。

最后还要感谢所有给予我帮助和支持的朋友们，谢谢大家。