OntoBase2.0 v3.0.0 Manual

Server

주식회사 리스트

Linked Data & Semantic web Technology

OntoBase2.0 Manual

LiST
Linked Data & Semantic web Technology

목차

Chapter 1. Introduction 2

1. OntoBase2.0 개요 2

1.1 시스템 개요 2

1.2 시스템 구성 2

1.3 시스템 상세 설명 3

2. OntoBase2.0 시스템 특징 3

2.1 시스템 특징 3

3. 기술 지원 5

3.1 기술 지원 5

4. 설치 구성 항목 6

4.1 항목 6

Chapter 2. Installation 8

1. OntoBase2.0 Installation 8

1.1 시스템 요구 사항 8

1.2 OntoBase2.0 Install 8

2. OntoBase2.0 Server의 시작 10

2.1 배포 디렉토리 구조 10

2.2 인증키 관리 11

2.3 서버의 시작과 정지 11

2.4 그외 제공 스크립트 13

3. OntoBase2.0 Manager 설치 방법 15

3.1 시스템 요구 사항 15

3.2 배포 디렉토리 구조 15

3.3 Manager의 설치 및 시작 16

4. OntoBase2.0 Client 설치 방법 19

4.1 배포 디렉토리 구조 19

4.2 Client 의 설치 및 시작 19

Chapter 3. OntoBase2.0 Server 21

1. OntoBase2.0 Server 개요 21

1.1 시스템 개요 21

1.2 시스템 구성 21

1.3 프레임워크 22

1.4 Service 22

1.5 Server Manager 25

1.6 Internal Database 25

1.7 Triple Store 26

2. Server Setting 29

2.1 Server 아이피 및 포트 변경하기 29

2.2 Server 실행 쓰레드 풀 개수 설정하기 29

2.3 모니터링 서비스의 메일 정보 설정하기 30

2.4 로그 서비스 정보 설정하기 30
2.5 기타 서버에 관한 설정들 31
3. Server Work Process 32
3.1 서비스의 생성 및 삭제 32
3.2 스토어의 생성 및 삭제 33
3.3 데이터 로딩 - 파일 빌드 35
3.4 데이터 로딩 - 수동 빌드 36
3.5 질의 처리 - Select 쿼리 37
3.6 질의 처리 - Ask 쿼리 37
3.7 질의 처리 - Construct 쿼리 38
3.8 질의 처리 - Describe 쿼리 38
Chapter 4. OntoBase2.0 Manager 40
1. OntoBase2.0 Manager 개요 40
1.1 시스템 개요 40
2. OntoBase2.0 Manager 구성 40
2.1 구성 40
3. OntoBase2.0 Manager 화면 구성 41
3.1 로그인 41
3.2 개요 42
3.3 저장소 45
3.4 데이터 49

3.5 SPARQL 61
3.6 관리도구 64
3.7 계정 66
Chapter 5. OntoBase2.0 Client 70
1. OntoBase2.0 Client 개요 70
1.1 시스템 개요 70
2. OntoBase2.0 Client 구성 70
2.1 클라이언트 API 라이브러리 70
2.2 API Document 71
2.3 사용 예제 71
3. 주요 라이브러리 71
3.1 StoreManager.java 71
3.2 ServerBuildManager.java 72
3.3 SparqlQuery.java 73
4. 지원 옵션 및 사용 예제 78
4.1 SPARQL : Basic Graph Patterns 79
4.2 SPARQL : Group Graph Patterns 80
4.3 SPARQL : Variable Constraints 81
4.4 SPARQL : Optional Pattern Matching 85
4.5 SPARQL : Matching Alternatives 87
4.6 SPARQL : Sequences and Modifiers – Order By 88

4.7 SPARQL : Sequences and Modifiers – Distinct 89

4.8 SPARQL : Sequences and Modifiers – Offset and Limit 90

4.9 SPARQL : RDF Datasets - Graph 91

4.10 ARQ Extension Option : All Variable 93

4.11 ARQ Extension Option : Count 94

4.12 ARQ Extension Option : Group By 95

4.13 ARQ Extension Option : Let 96

4.14 ARQ Extension Option : Sub Query 97

4.15 ARQ Extension Option : Filter Functions 98

4.16 ARQ Extension Option : Filter Forms - IF 102

4.17 ARQ Extension Option : Filter Forms - COALESCE 103

4.18 ARQ Extension Option : Property Paths 104

Chapter 6. TroubleShooting 109

1. OntoBase2.0 Server 109

1.1 Q-1: 서버 구동 시 메모리가 부족합니다. (OutOfMemory 에러) 109

1.2 Q-2: 데이터 빌드 중에 서버 강제 종료 후 에러가 발생합니다. 109

2. OntoBase2.0 Manager 110

2.1 Q-1: 관리도구 로그인 페이지에 접속 후 다음과 같은 메시지가 나타납니다. 110

2.2 Q-2: 관리도구의 서버 메뉴의 Synchronization 항목에서 서비스나 스토어 시작 / 종료시 java.lang.NullPointerException 오류가 발생하는 경우 111

부록 114

A. Rule File 114

B. 색 인 134

Chapter 1

Introduction

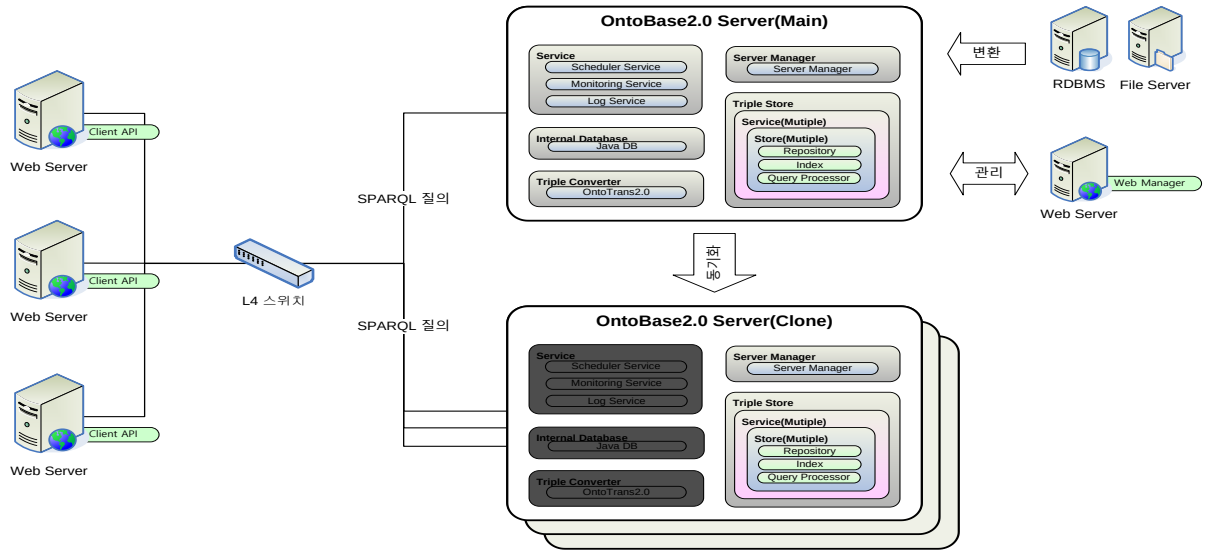
Chapter 1. Introduction

1. OntoBase2.0 개요

1.1 시스템 개요

OntoBase2.0 은 대용량 트리플 데이터의 저장, 처리, 질의 및 관리 기능을 통합한 RDF/OWL 온톨로지 레파지토리 (Ontology Repository)이다.

1.2 시스템 구성



[그림 1.1 OntoBase2.0 시스템 구성도]

1.3 시스템 상세 설명

1) OntoBase2.0 Server

메시지 기반의 프레임워크를 기반으로 대용량의 트리플 데이터를 저장하고 빠른 검색을 위한 색인을 구성하며 쿼리에 대한 요청을 처리하는 트리플 레파지토리로서 유연성과 안정성을 고려하여 다양한 부가 기능을 갖춘 서버이다.

2) OntoBase2.0 Manager

서버에 대한 관리 역할을 수행하는 관리자를 위한 웹 관리도구로서 서버의 환경 설정 등에 대한 서버의 다양한 기능을 손쉽게 실행 가능하도록 지원한다.

3) OntoBase2.0 Client

질의 명령 등을 수행할 수 있는 클라이언트 API 라이브러리로서 서버에 데이터를 추가/삭제 및 질의를 생성하고 요청하며, 결과를 받아 처리할 수 있는 로직 등이 포함된다.

2. OntoBase2.0 시스템 특징

2.1 시스템 특징

1) 다중 분산 시스템
   - 가) 안정성과 고가용성을 고려한 분산구조
   - 나) 호환성 및 확장성을 고려한 메시지 기반의 프레임워크 적용
   - 다) 효율성과 안정성으로 고려한 Non-Blocking 방식의 네트워크 서버 적용
   - 라) 로드 밸런싱을 고려한 시스템 구조 지원

마) 네트워크 처리 속도 향상을 위해 pooling 기법을 적용

바) 서버들의 관리와 모니터링 등을 위한 자체 내장 DB 를 가지는 웹 관리도구 지원

2) 대용량 데이터 기반 시스템

가) 대용량을 위한 파일 기반 시스템

나) 다양한 플랫폼을 고려한 파일 분할 기법 적용

다) 트리플 데이터의 특성을 고려한 디스크의 용량을 최소화하는 최적화된 자료 저장구조 적용

라) 대용량 데이터의 고속 벌크 로딩 모듈 지원

3) 저장 및 관리에 최적화된 구조

가) 최소한의 디스크 I/O 를 위해서 메모리 캐싱 기법

나) 속도 및 효율성을 고려한 Block 단위의 데이터 구조

다) 파일 로딩 및 변환기를 통한 로딩 등의 다양한 로딩 모듈 지원

라) 실시간 로딩 모듈 지원

4) 빠른 질의 모듈

가) 빠른 쿼리를 위한 최적화된 색인구조 적용

나) 다양한 멀티 쿼리에 대한 쿼리 옵티마이즈 로직 적용

다) 비동기식 I/O 처리

라) 커넥션 풀링 사용

마) 정교하게 최적화된 Low 레벨의 데이터 I/0 모듈 적용

5) 국제 표준을 준수

가) W3C 의 RDF, RDFS, OWL 등 다양한 표준 지원

나) 저장된 트리플의 질의 처리를 위해 SPARQL 지원

6) 다양한 부가 기능 지원

가) 관리자를 위한 통합 웹 관리 모듈 제공

나) 다양한 로그 관리 및 장애 관리 모듈 제공

다) 데이터 복구 모듈 지원

3. 기술 지원

3.1 기술 지원

1) 기술지원

가) 제품에 대한 불편사항이나 궁금한 사항에 대해 당사는 OntoBase2.0 홈페이지를 운영하고 있으며, 이곳을 통해 OntoBase2.0 에 대한 기술지원을 요청할 수 있습니다.

나) 제품 공급자 정보

- 공급자명: ㈜리스트
- 공급자 주소: 서울시 영등포구 버드나루로 19 길 3 양용빌딩 901 호
- 공급자 전화번호: (02) 2632-5133
- 제품 홈페이지: http://li-st.com/products/ontobase.jsp

4. 설치 구성 항목

4.1 항목

1) OntoBase2.0 솔루션은 사용자에게 다음과 같은 패키지 형태로 제공됩니다.

   가) OntoBase 2.0 설치 CD 1 부 (설치 파일 및 매뉴얼 전자문서)

   나) 사용자 매뉴얼 1 부

   다) 제품 인증키 1 부 (상용 제품 사용자인 경우 첨부)

Chapter 2

Installation

Chapter 2. Installation

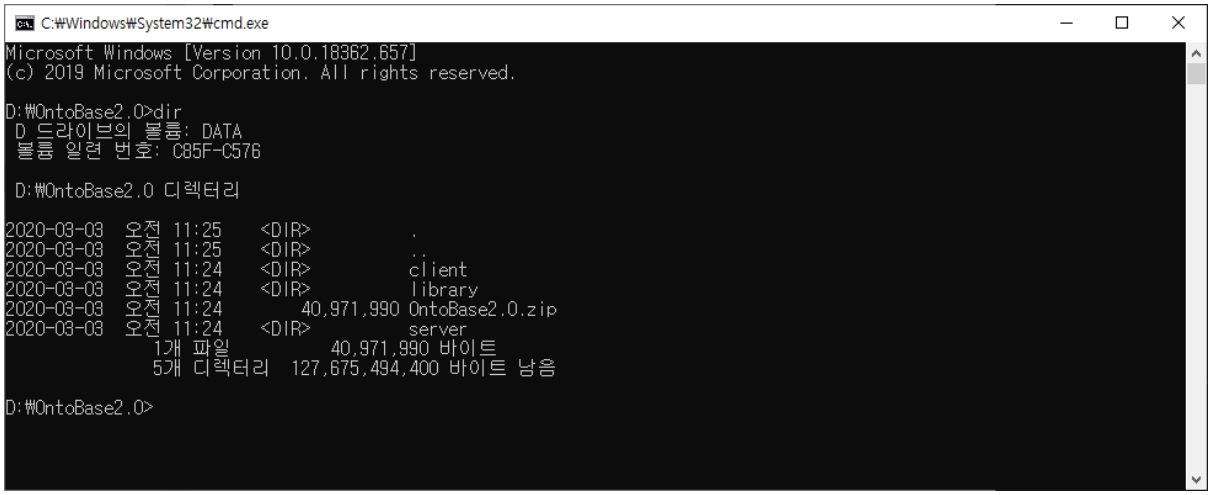
1. OntoBase2.0 Installation

1.1 시스템 요구 사항

1) 시스템 요구 사항
   가) 실행 언어: JRE 8.0 이상
   나) 지원 플랫폼: Java 가 지원하는 모든 플랫폼(Windows MS 계열 및 Linux, Solaris, IBM, HP-UX 등의 Unix 계열 포함)
   다) 최소 RAM 1G 이상

1.2 OntoBase2.0 Install

1) Windows MS 계열
   가) 실행언어인 자바 JRE 8.0 이상을 설치한다. (java -version 명령어를 통해 정상적으로 설치되었는지 확인한다.)
   나) 동봉되는 CD 의 windows 용 설치파일을 설치하고자 하는 디렉토리에 복사한다. (OntoBase2.0.zip)
   다) 해당 디렉토리에 압축을 해제한다.



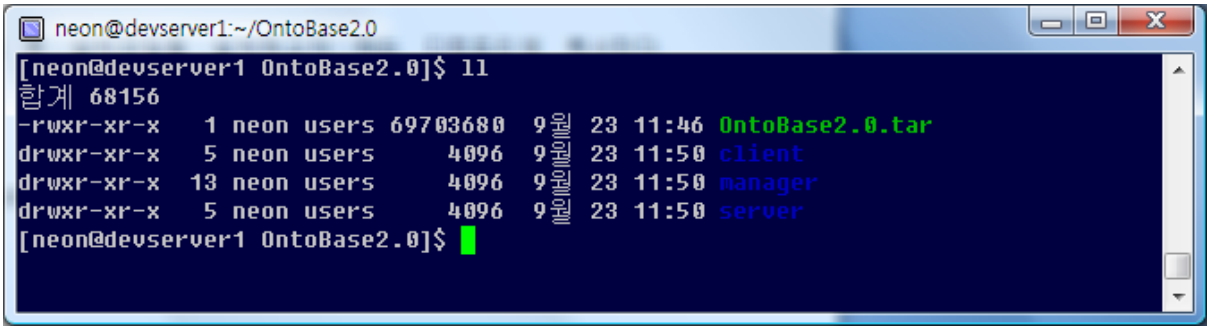
[그림 2.1 Windows 계열의 OntoBase2.0 설치 완료 화면]

2) Linux, Solaris, IBM, HP-UX 등의 Unix 계열

가) 실행언어인 자바 JRE 8.0 이상을 설치한다. (java -version 명령어를 통해 정상적으로 설치되었는지 확인한다.)

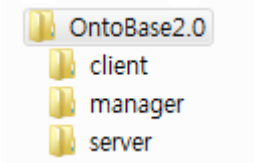
나) 동봉되는 CD 의 linux 용 설치파일을 설치하고자 하는 디렉토리에 복사한다. (OntoBase2.0.tar)

다) 압축 명령어 tar –xvf OntoBase2.0.tar 를 이용해 압축 해제한다.



[그림 2.2 Unix 계열의 OntoBase2.0 설치 완료 화면]

3) 설치가 완료되면 다음과 같은 폴더가 생성된다.



[그림 2.3 설치완료 후 디렉토리 구조]

가) Client – OntoBase2.0 Client

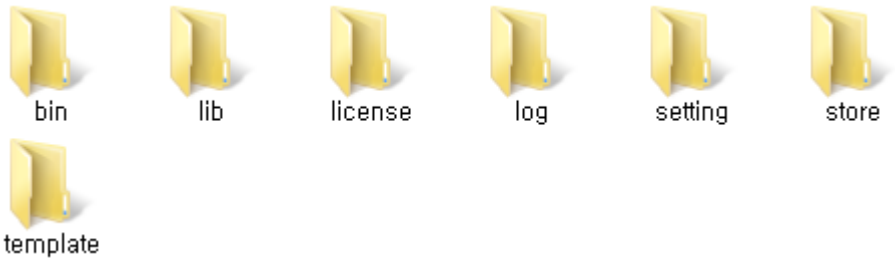
나) Manager – OntoBase2.0 Manager

다) Server – OntoBase2.0 Server

2. OntoBase2.0 Server의 시작

2.1 배포 디렉토리 구조

1) 디렉토리 구조



[그림 2.4 OntoBase2.0 Server 배포 디렉토리 구조]

2) 디렉토리 설명

가) bin - 실행 스크립트 및 내부 데이터베이스 디렉토리

나) lib - 핵심 라이브러리 디렉토리

다) License – license 정보 및 제품 인증키(authentication.txt) 디렉토리

라) log - 시스템 로그 파일 디렉토리

마) setting - 시스템 설정 디렉토리

바) store - 실제 데이터 저장소 디렉토리

사) template – 서비스 시 사용되는 템플릿 파일

2.2 인증키 관리

1) 인증키 적용 방법

가) License 폴더의 authentication.txt 파일이 제품에 등록된 라이선스 파일입니다.

2.3 서버의 시작과 정지

1) 서버의 시작

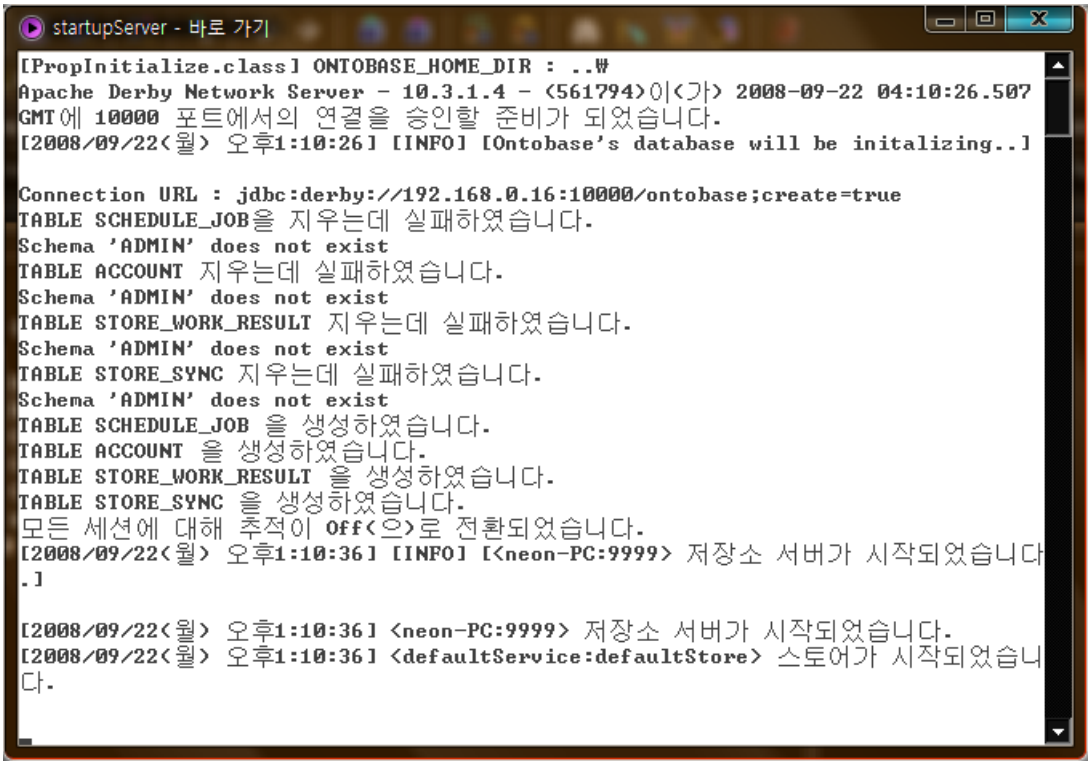
가) $SERVER_HOME₩bin 디렉토리 아래의 startup.bat(startup.sh)파일을 실행한다.

나) 서버가 실행되면 설정 정보로부터 정보를 읽어 구동한다.

다) 디폴트로 서버의 아이피는 localhost, 서버의 포트는 9999 번으로 셋팅 되어 있다.

라) 디폴트로 defaultService 라는 이름의 서비스와 defaultStore 라는 이름의 스토어가 셋팅되어 있다.

마) 서버가 구동하면서 자동으로 Internal Database 도 같이 구동된다. 내부 데이터베이스는 $SERVER_HOME₩bin 디렉토리 아래에 ontobase 라는 이름으로 생성된다.



[그림 2.5 OntoBase2.0 Server 의 시작 화면]

바) 서버 시작시 초기화를 위한 약간의 지연시간이 존재합니다. 이와 같은 지연시간 때문에 관리도구의 서버 메뉴항목에서는 서버의 상태가 stopped 로 표시될 수 도 있다. 저장소 서버 시작은 등록된 모든 스토어가 시작되어야만 관리도구에서 올바른 정보가 나타난다. 관리자는 이와 같은 이유로 서버가 완전히 구동되는 것을 확인 후 관리도구를 사용해야 한다.

2) 서버의 중지

가) $SERVER_HOME₩bin 디렉토리 아래의 shutdown.bat(shutdown.sh)파일을 실행한다.

나) 서버가 중지되면 서버는 자동으로 메모리에 캐시 되어 있는 작업들을 저장하고 종료한다.

2.4 그외 제공 스크립트

1) registFileBuild.bat(registFileBuild.sh)

가) 설명: 트리플 데이터를 서버로 Build 한다.

나) 설정

- ID: 접속 아이디(디폴트 제공 아이디 – admin)
- PWD: 접속 패스워드(디폴트 패스워드 - admin@#$)
- IP: 서버 아이피
- PORT: 서버 포트
- SERVICE: 대상 서비스 명
- STORE: 대상 스토어 명
- DIR: 로딩할 트리플 데이터 디렉토리(디렉토리 내의 모든 파일을 로딩한다.)
- RUNTYPE: Build 실행 타입

  [1]: DIR 에 존재하는 파일들을(서버가 동일한 위치) Bulk 빌드하도록 서버에 명령을 등록한다. Bulk 빌드는 기존의 모든 데이터를 삭제하고 빌드한다.

  [2]: DIR 에 존재하는 파일들을(서버가 동일한 위치) 추가 빌드하도록 서버에 명령을 등록한다. 추가 빌드는 기존의 데이터를 유지하고 데이터를 추가한다.

  [3]: DIR 에 존재하는 파일들을(서버가 동일한 위치) 삭제 빌드하도록 서버에 명령을 등록한다. 삭제 빌드는 기존의 데이터를 유지하고 데이터를 삭제한다.

  [4]: DIR 에 존재하는 파일들을(서버가 다른 위치) 서버로 전송하여 Bulk 빌드하도록 서버에 명령을 등록한다. Bulk 빌드는 기존의 모든 데이터를 삭제하고 빌드한다.

  [5]: DIR 에 존재하는 파일들을(서버가 다른 위치) 서버로 전송하여 추가 빌드하도록 서버에 명령을 등록한다. 추가 빌드는 기존의 데이터를 유지하고 데이터를 추가한다.

  [6]: DIR 에 존재하는 파일들을(서버가 다른 위치) 서버로 전송하여 삭제 빌드하도록 서버에 명령을 등록한다. 삭제 빌드는 기존의 데이터를 유지하고 데이터를 삭제한다.

2) clientConsole.bat(clientConsole.sh)

가) 설명: OntoBase2.0 접속 및 Sparql 질의 및 접속 테스트 할 수 있다.

가) 설정

- IP: 서버 아이피
- PORT: 서버 포트

3) queryTest.bat(queryTest.sh)

가) 설명: 질의 테스트를 할 수 있다.

나) 설정

- IP: 서버 아이피
- PORT: 서버 포트
- SERVICE: 대상 서비스 명
- FETCHSIZE: 페치 개수
- QUERYTYPE: 질의 타입

  [1]: SPARQL SELECT 쿼리.

  [2]: SPARQL ASK 쿼리.

  [3]: SPARQL DESCRIBE 쿼리.

  [4]: SPARQL CONSTRUCT 쿼리.
- QUERYFILE: 실제 쿼리가 들어가 있는 파일 명. 파일 내에 질의할 SPARQL 쿼리를 작성한다.

3. OntoBase2.0 Manager 설치 방법

3.1 시스템 요구 사항

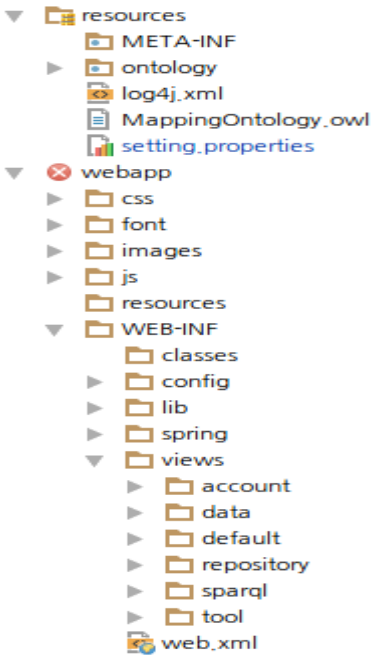
1) 시스템 요구 사항

가) 실행 언어: JRE 8.0 이상

나) 웹 서버, J2EE 컨테이너에 설치

3.2 배포 디렉토리 구조

1) 디렉토리 구조



[그림 2.6 OntoBase2.0 Manager 배포 디렉토리 구조]

2) 디렉토리 설명

가) resource - 시스템 관련 리소스 파일 관리 디렉토리

나) css – css 관련 디렉토리

다) font – 폰트 관련 디렉토리

라) images – 이미지 관련 디렉토리

마) js – 자바 스크립트 파일이 들어있는 디렉토리

바) classes – 컴파일된 클래스파일 관리 및 스프링 설정 관련 디렉토리

사) config – 시스템에서 사용될 타일즈 설정 파일 관련 디렉토리

아) lib – 사용될 라이브러리 디렉토리

자) spring – 스프링 콘텍스트 설정 파일 관리 디렉토리

차) views – 뷰 관련 디렉토리

3.3 Manager의 설치 및 시작

1) 설치 (Context 등록)

A. 서비스중인 서블릿 엔진의 Context 에 OntoBase2.0 manager 를 등록한다.

B. Tomcat 5.5 설정 예시

① %TOMCAT_HOME%₩conf₩Catalina₩localhost 컨텍스트로 사용할 이름으로 새로운 파일을 생성한다. (예.mgr.xml)

Mgr.xml 의 파일 내용

*<Context **docBase="% ONTOBASE_HOME%₩manager"** privileged="true"*
antiResourceLocking="false" antiJARLocking="false"/>

② docBase 를 관리도구 설치폴더인 manager 의 절대경로를 입력후에 저장하게 되면 파일명인 mgr 로 새로운 컨텍스트가 등록되게 된다.

2) 설정

A. $MANAGER_HOME/WEB-INF/classes/setting.properties

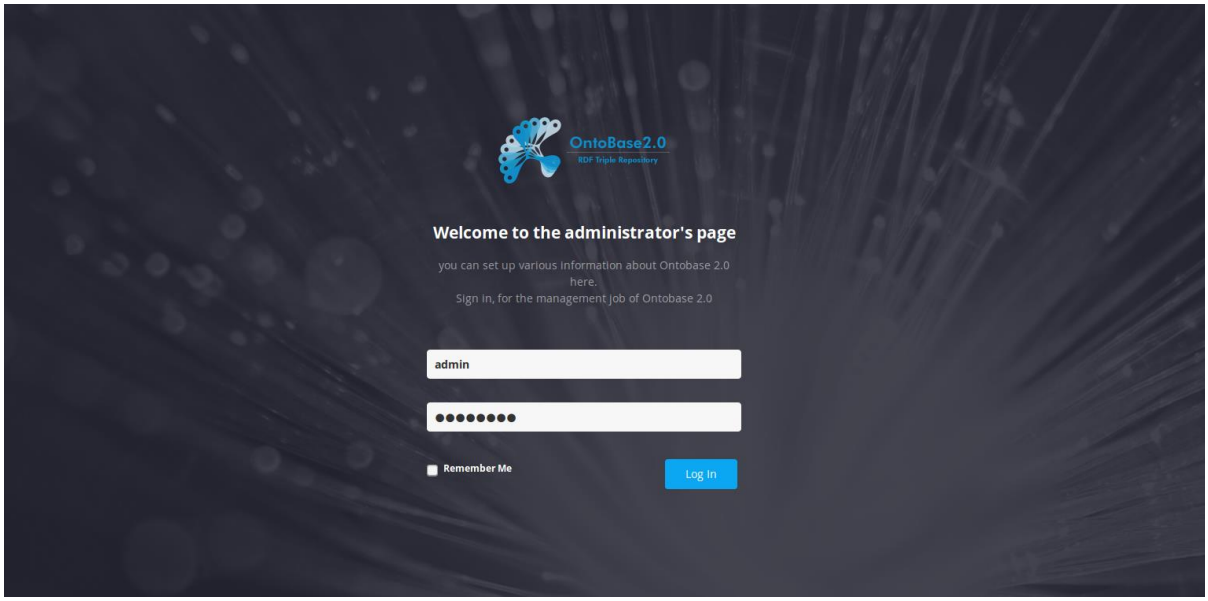
① 관리도구에서 사용할 정보를 점검한다. 설명 중 (파일)이라고 명시한 프로퍼티 값은 반드시 파일명까지 명시되어야 한다.

i. database.user : 데이터 베이스 접속 유저 명 (default : admin)
ii. database.pwd : 데이터 베이스 접속 암호 (default : admin)
iii. database.connect.dbname : 데이터 베이스 접속 명. (default : ontobase)
iv. database.connect.ip : 데이터 베이스 접속 주소
v. database.connect.port : 데이터 베이스 접속 포트
vi. system.alias: 온투베이스 서버 어라이어스
vii. system.ip: 온투베이스 접속 아이피
viii. system.port: 온투베이스 접속 포트
ix. system.manager.port : 온토베이 스매니 저접속 포트 (default :9995)
x. mappingOntology : 트리플 변환 수행 시 사용되는 매핑 온톨로지 파일 절대경로 (파일)
xi. settingFile : 관리도구에서 사용될 정보를 기록한 프로퍼티 파일 (파일)
xii. tmpSaveDir: 파일 업다운로드 관련 임시 디렉트리 경로
xiii. pubFileDir: 발행관리 수행 시 발행관리된 트리플 파일을 저장하는 디렉토리 경로
xiv. syncDir : 발행관리에 사용되는 파일 경로로서 파일이름은 result.ttl 로 지정되어야 함 (파일)
xv. transFileDir : 변환관리를 거쳐 변환된 트리플 데이터를 저장하는 디렉토리 절대 경로
xvi. converter.path : 변환기 절대 경로
xvii. pageSize : 페이징 처리 시 사용되는 페이지 사이즈
xviii. pagingSize : 페이징 처리 시 사용되는 페이징 사이즈

3) 시작

A. JSP 서블릿 엔진에 등록된 주소로 접속을 시도한다.
B. 접속 예) http://localhost:8080/mgr/index.onto

C. 아래와 같은 로그인 화면이 보이면 기본적으로 제공되는 계정인 ID: admin, Password : admin@#$ 을 입력하여 관리도구를 이용할 수 있다

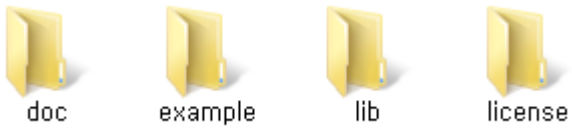


[그림 2.7 OntoBase2.0 Manager 관리도구 초기화면]

4. OntoBase2.0 Client 설치 방법

4.1 배포 디렉토리 구조

1) 디렉토리 구조



[그림 2.8 OntoBase2.0 Client 배포 디렉토리 구조]

2) 디렉토리 설명

가) doc - 클라이언트 API Document 들이 포함되어 있는 디렉토리

나) example - 클라이언트 API 를 사용하여 구현된 예제 파일들이 들어있다.

다) lib - Java 로 구현된 클라이언트 API

라) License - 프로그램 라이선스 관련 파일

4.2 Client 의 설치 및 시작

1) 사용하고자 하는 애플리케이션에 lib 디렉토리의 클라이언트 모든 라이브러리들을 임포트해서 사용한다.
2) 사용 방법은 Chapter5 의 클라이언트 부분을 참조한다.

Chapter 3

OntoBase2.0 Server

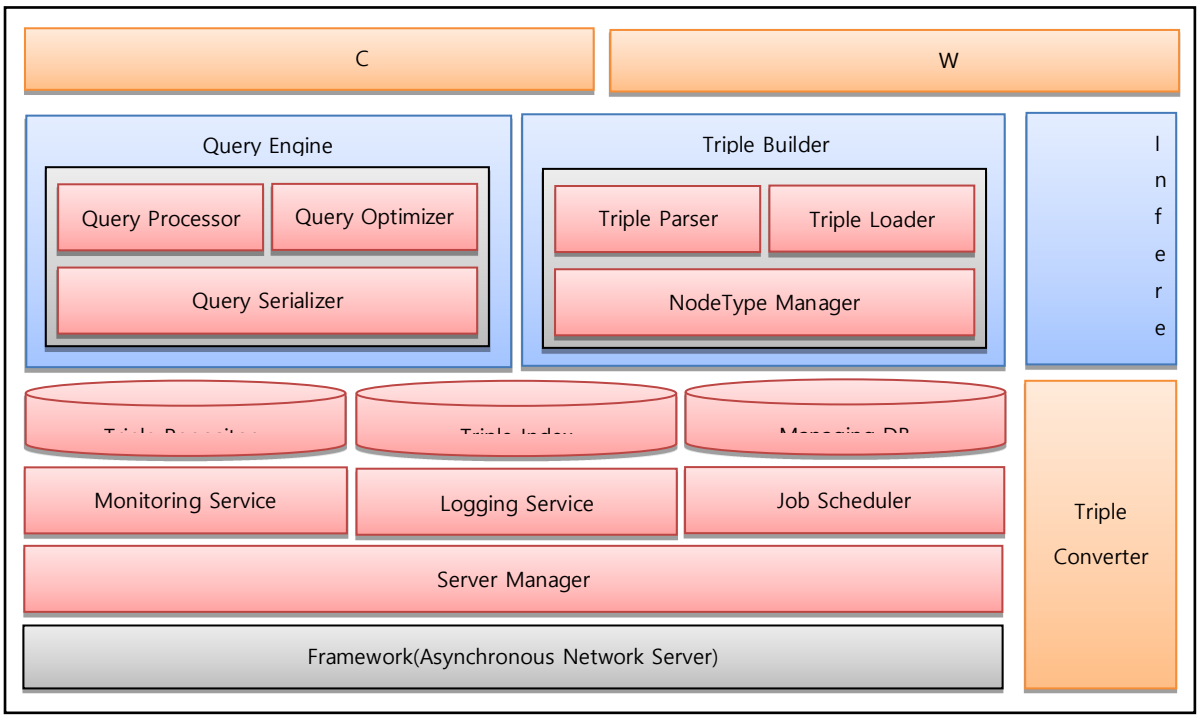
Chapter 3. OntoBase2.0 Server

1. OntoBase2.0 Server 개요

1.1 시스템 개요

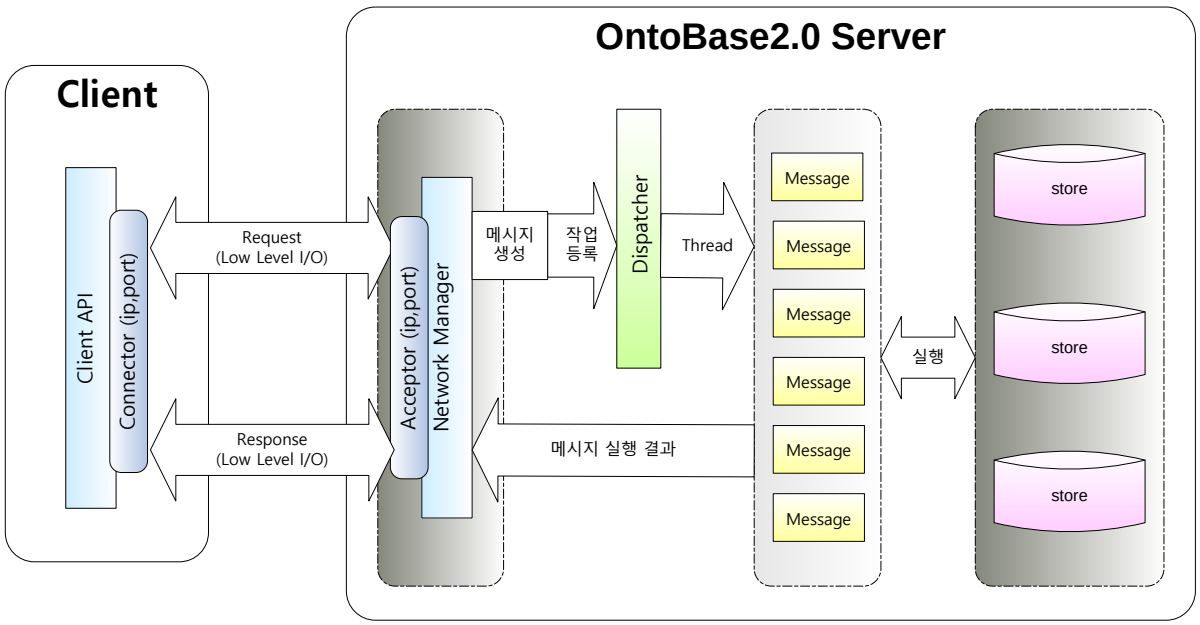
OntoBase2.0 Server 는 OntoBase2.0 의 서버 모듈로서 메시지 기반의 프레임워크를 기반으로 대용량의 트리플 데이터의 저장 및 질의 처리를 수행하는 트리플 레파지토리로서 유연성과 안정성을 고려하여 다양한 부가 기능을 갖춘 서버이다.

1.2 시스템 구성



[그림 3.1 OntoBase2.0 Server 시스템 구성도]

1.3 프레임워크



[그림 3.2 메시지 기반 프레임워크의 구조]

1) 메시지 기반 프레임워크

가) 개요 - 최적화를 위해서 Low 레벨의 I/O 를 적용한 메시지 기반의 프레임워크 채택

나) 메시지 기반 프레임워크의 구조

1.4 Service

1) Service 의 개요

가) 서버의 다양한 부가 기능을 제공하는 모듈로서 서버내의 작업 관련 프로세스의 등록 및 실행 서버들의 상태를 모니터링하고 에러 발생 시 처리 등의 컴포넌트가 존재한다.

2) Service 의 구성

가) Scheduler Service - 서버에 관련된 작업들을 등록하고 실행 시켜주는 스케줄러 서비스 관련 컴포넌트

나) Monitoring Service - 서버의 상태를 감시하고 리포팅 해주는 모니터링 서비스 관련 컴포넌트

다) Log Service - 서버의 상황에 대한 다양한 정책을 적용한 로그 서비스 관련 컴포넌트

3) Scheduler Service

가) 스케줄러에 등록된 정보는 데이터베이스에 의해 관리되어 서버가 재 시작 했을 경우 기존 등록된 스케줄링 작업이 유효하다

나) 각 스케줄 작업은 실행 상태를 데이터베이스에 업데이트함으로써 실시간 으로 상태를 감지한다.

다) 일시적인 단발성 작업이 스케줄러에 등록되었을 경우 중복 작업 여부를 판단하여 등록한다.

라) 이 스케줄러의 실제 실행은 웹 관리 도구를 통해서 등록이 가능하고 자동으로 실행된다.

4) Monitoring Service

가) 일정시간 간격으로 등록된 서버의 상태를 확인하여 서버들의 동작 상태를 체크하고 상황을 관리한다.

나) 일정시간 간격으로 등록된 서비스들에게 단순 질의를 통해 서비스들의 동작 상태를 체크하고 상황을 관리한다.

다) 중요한 오류상황 혹은 중요한 정보라고 판단되는 경우 관련 정보를 로그로 남긴다.

라) 메일 전송 기능이 활성화 되어 있다면 오류 발생 시에 등록된 관리자에게 메일로 관련 사항을 전송한다.

마) 모니터링 서비스에 대한 상세 환경 설정은 $SERVER_HOME/setting/system.properties 에서 가능하다. (Server Setting 부분 참조)

5) Log Service

가) 로그 서비스는 다양한 형태의 서버 관련 로그를 제공한다.

나) 기본 로그의 레벨 설정은 시스템 설정 파일에서 설정이 가능하다.

다) 로그의 경우에 시스템 로그 및 각 스토어 내의 로그로 구분된다. 시스템 로그의 경우는 전체 서버 시스템에 대한 로그를 남기고 스토어 내에서는 각각 쿼리에 관련된 로그, 스토어의 상태에 관련된 로그 등으로 구분된다.

A. 시스템 로그 - $SERVER_HOME/log 디렉토리에 위치

① system.log - 저장소 서버에서 발생하는 각종 정보 및 오류 정보를 기록

② service.log - 저장소 서버에서 발생하는 서비스에 대한 info 를 기록

③ consoleManager.log - 관리서버에서 발생하는 정보 및 오류 정보를 기록

B. 스토어 로그 - $SERVER_HOME/store 디렉토리에 위치

① $store_name.query.log - 스토어에 질의된 쿼리를 기록

② $service_name.$store_name.log - 스토어 서비스 중 발생하는 정보 및 오류 정보를 기록

라) 모든 로그 파일은 하루 단위로 백업되어 관리된다.

마) 오래된 로그 파일의 경우는 자동으로 삭제되며, 삭제 기간의 경우는 시스템 설정에서 설정 가능하며 기본 유지 기간은 30 일이다.

바) 로그 서비스에 대한 상세 환경 설정은 $SERVER_HOME/setting/system.properties 에서 가능하다. (Server Setting 부분 참조)

1.5 Server Manager

1) Server Manager 의 개요
   가) Server Manager 는 OntoBase2.0 저장소 서버를 관리 및 감시하고 관리도구와 연계되어 사용자 명령을 저장소 서버로 전달하는 역할을 담당하는 관리 기능의 모듈이다.
2) Server Manager 의 기능
   가) 저장소 서버를 시작 혹은 종료하는 것을 담당한다.
   나) 저장소 서버의 서비스 또는 스토어의 동작을 관리한다.
   다) 로그, 스케줄러, 모니터링과 같은 서버의 부가 서비스들을 관리한다.

1.6 Internal Database

1) Internal Database 의 개요
   가) 개요 - 웹 관리 도구 및 서버내의 작업 등록 등에서 사용되는 내부 데이터 베이스 모듈
   나) 서버가 구동되면 자동으로 내부 데이터 베이스를 시작시키며 내부 데이터 베이스의 생성 위치는 $SERVER_HOME/bin 디렉토리에 생성된다.
2) Internal Database 의 생성 테이블
   가) Account - 계정 정보 테이블
   나) Schedule_Job - 스케줄 작업 정보 테이블
   다) Store_Work_Result - 스토어의 작업 결과 테이블
3) Internal Database 의 초기화
   가) 내부 데이터 베이스는 자동으로 서버가 구동될 때, 테이블의 존재 유무를 체크해서 없다면 생성해준다.
   나) 만약, 수동으로 데이터 베이스를 초기화해야 할 필요가 있는 경우는 $SERVER_HOME/bin 디렉토리의 initTable 파일을 실행한다.

1.7 Triple Store

1) Triple Store 의 개요

가) 개요 - 실제 트리플을 저장하고 처리하는 핵심 모듈. 원본 트리플 데이터를 저장하고 빠른 검색을 위한 색인을 구성하며 쿼리에 대한 요청을 처리하는 모듈 등으로 구성된다.

나) 서비스 및 스토어의 생성 및 구성은 관리도구를 통해서 설정 가능하다.

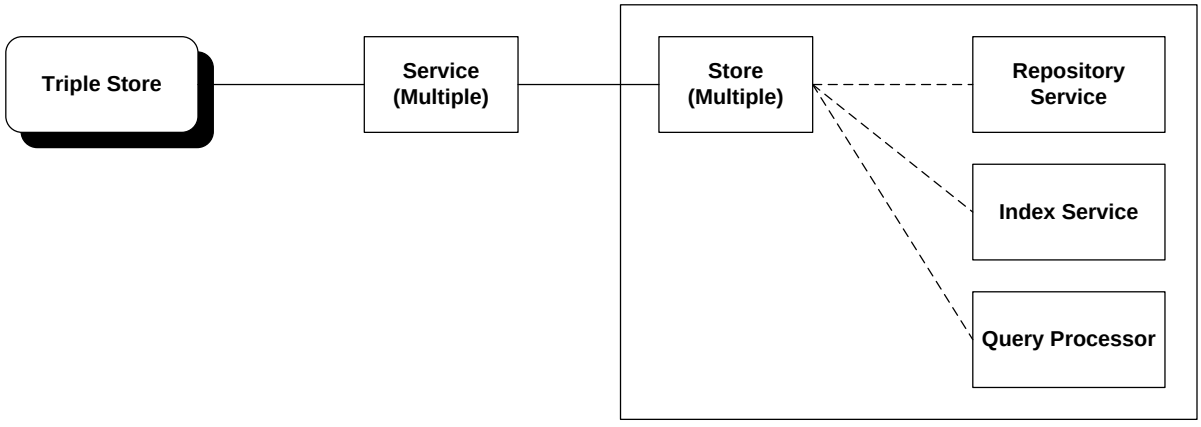
2) Triple Store 의 구조

가) Triple Store 는 여러 개의 Store 로 구성 가능하다.

나) 하나의 Store 는 Triple Store 의 최소 처리 단위이며 각 Graph 에 대한 Repository 와 Index 및 Query Processor 구조가 생성되며 선택적으로 변환 모듈을 연동할 수 있다.

다) 변환모듈은 기존 레거시 DB 혹은 다른 데이터 소스로부터 Triple 데이터로 변환하여 자동으로 저장소에 넣는 모듈을 의미하며, 이 변환모듈에서는 전체, 추가 변환, 추론 등을 지원한다. 이 변환모듈의 기본 셋팅 정보(모델)가 변경이 되면 해당 Store 는 초기화 된다.

라) Store 에 변환모듈을 연동하지 않으면 파일 형태의 Triple 데이터를 사용자가 직접 넣을 수 있으며, 전체 빌드(Bulk Build)/추가 빌드 등을 지원한다.



[그림 3.4 Triple Store 의 논리적 구조]

3) Repository Service

가) Repository Service 의 개요

A. Repository Service 는 Triple 에 대한 모든 정보를 효율적으로 저장하여 관리 하기 위한 저장소이다.

B. 이 저장소는 각 트리플을 저장하는 Triple Repository 와 각 노드의 원본 데이터들을 저장하는 Node Repository, 프리픽스 정보를 저장하는 Prefix Mapping 등으로 구성된다. 각 트리플 데이터들은 중복 처리, 프리픽스 적용 등의 처리를 통해 최적화되어 저장된다.

나) Repository Service 의 구성

A. Triple Repository – 트리플 데이터를 저장하고 관리한다.

B. Node Repository – 노드 데이터를 저장하고 관리한다.

C. Prefix Mapping - 프리픽스 정보를 저장하고 관리한다.

다) Repository Service 의 특징

A. 대용량 처리가 가능하도록 모두 File 로 저장되어 분할 관리된다.

B. 속도 향상 및 자원 사용의 효율성을 위한 최적화된 캐시 구조를 사용하여 메모리 사용을 제어한다.

C. 각 데이터의 형태에 맞게 최적화된 자료구조를 사용한다.

D. 중복 처리 및 프리픽스 적용 들을 통해 최소의 디스크 용량을 위한 구조를 사용한다.

E. 각 데이터의 형태에 맞게 최적화된 자료구조를 사용한다.

F. 변동 정보의 저장을 통해서 데이터 복구 로직을 지원한다.

G. 데이터의 변동 적용을 위한 추가 빌드 로직을 지원한다.

H. 삭제는 빠른 처리를 위해 물리적으로 삭제하지 않고 논리적 삭제로 처리하며, 이에 대해서 별도의 로직을 지원한다.

I. 초기 데이터의 빠른 빌드를 위한 Bulk 빌드 로직을 지원한다.

4) Index Service

가) Index Service 의 개요

A. Index Service 는 트리플에 대한 효과적인 검색을 위한 색인을 구성하여 관리하기 위한 저장소이다.

B. 이 저장소는 노드 아이디의 구성 형태에 따라 크게 SPO Index 와 Property Index 로 구성되며, 각 인덱스들은 트리플을 구성하는 아이디들의 Full Path 로 미리 정렬되어 구성된다.

나) Index Service 의 구성

A. SPO Index - Subject ID, Predicate ID, Object ID 의 3 아이디의 조합으로 아이디의 조합으로 구성된 인덱스

B. Property Index - Predicate ID 로 구성된 인덱스

다) Index Service 의 특징

A. 대용량 처리가 가능하도록 모두 File 로 저장되어 분할 관리된다.

B. 속도 향상 및 자원 사용의 효율성을 위한 최적화된 캐시 구조를 사용하여 메모리 사용을 제어한다.

C. 데이터의 변동 적용을 위한 추가 빌드 로직를 지원한다.

D. 최적화된 색인의 구성을 통한 빠르고 효과적인 검색을 보장한다.

E. 초기 데이터의 빠른 빌드를 위한 Bulk 빌드 로직을 지원한다.

5) Query Processor

가) Query Processor 의 개요

A. Query Processor 는 엔진에서 제공되는 Client API 를 통해 요청되는 SPARQL 쿼리를 처리하고 처리된 결과를 전송하기 위한 쿼리 처리기를 말한다.

B. Query Processor 에는 크게 Query Parser, Query Optimizer, Query Serializer 로 구성된다.

나) Query Processor 의 구성

A. Query Parser - 요청 쿼리를 파싱하고 엔진에서 처리 가능한 모듈로 변환하는 모듈

B. Query Optimizer - 복잡한 요청 쿼리에 대해서 최적의 성능을 위해서 쿼리를 재배치하고 최적화하는 모듈

C. Query Serializer - 요청 쿼리에 대한 빠른 전송을 위한 low 레벨의 최적화된 I/O 모듈

다) Query Processor 의 특징

A. 쿼리의 처리를 위한 언어로서 SPARQL 을 사용한다.

B. SPARQL Spec 을 준수하여 정확한 처리를 위해서 오픈 소스인 ARQ 를 이용한다.

C. Multiple Triple Pattern 의 처리를 위해서 최적화 처리 로직을 적용하여 결과를 처리한다.

D. 결과의 빠른 처리를 위해서 Low 레벨의 최적화된 통신 모듈을 사용하여 처리한다.

2. Server Setting

2.1 Server 아이피 및 포트 변경하기

1) $SERVER_HOME₩setting 디렉토리 아래의 system.properties 파일을 수정하고 서버를 재시작 한다.
2) System.properties 파일의 관련 프로퍼티
   가) system.ip - 서버가 바인드할 아이피 주소
   나) system.port - 저장소 서버의 포트 번호
   다) system.manager.port - 관리서버의 포트번호

2.2 Server 실행 쓰레드 풀 및 쿼리 처리 정보 설정하기

1) $SERVER_HOME₩setting 디렉토리 아래의 system.properties 파일을 수정하고 서버를 재시작한다.
2) system.properties 파일의 관련 프로퍼티
   가) system.util.threadpool.size - 서버에서 사용할 쓰레드 풀의 개수 (디폴트는 100)

나) system.util.threadpool.maxsize – 서버에서 사용할 최대 쓰레드 풀의 크기

다) system.util.query.timeout – 디폴트 쿼리 타임아웃 타임(클라이언트에서 타임아웃을 설정할 경우는 우선 적용된다.)

라) system.util.query.log – 로그에 쿼리 출력 여부

2.3 모니터링 서비스의 메일 정보 설정하기

1) 이 서비스를 이용하기 위해서는 SMTP 를 지원하는 메일 계정서비스가 있어야만 사용이 가능하다.
2) $SERVER_HOME₩setting 디렉토리 아래의 system.properties 파일을 수정하고 재시작한다.
3) system.properties 파일의 관련 프로퍼티
   가) report.mail.use – 메일 모니터링 서비스 사용 유무로 디폴트는 false (사용: true)
   나) report.mail.smtp.host - 메일을 보내기 위한 SMTP 호스트 주소
   다) report.mail.smtp.auth.user - SMTP 계정 정보
   라) report.mail.smtp.auth.pwd - SMTP 의 계정 암호
   마) report.mail.from.email - 메일정보를 받을 관리자의 메일주소
   바) report.mail.to.email - 메일정보를 전송할 사람의 메일주소

2.4 로그 서비스 정보 설정하기

1) $SERVER_HOME₩setting 디렉토리 아래의 system.properties 파일을 수정하고 재시작한다.
2) system.properties 파일의 관련 프로퍼티
   가) system.log.system.level - 시스템 단위 로그의 출력 레벨 (DEBUG, INFO, WARN, ERROR, FATAL 의 5 레벨이 존재한다.)
   나) system.log.service.lever - 서비스 단위 로그의 출력 레벨 (DEBUG, INFO, WARN, ERROR, FATAL 의 5 레벨이 존재한다.)

2.5 기타 서버에 관한 설정들

1) $SERVER_HOME/setting/system.properties

가) 시스템 정보 및 메인 서버 정보

A. system.alias : 현재 서버가 사용하게될 별칭을 입력한다.

B. system.main : 현재 서버가 메인서버로서 동작할지 여부를 결정한다. (true 이면 메인서버로 동작한다.)

C. system.ip : 현재 서버가 바인드할 서버 주소를 입력한다.

D. system..port : 저장소 서버가 바인드할 포트 번호를 입력한다.

E. system.manager.port : 관리 서버가 바인드할 포트 번호를 입력한다..

F. system.log.system.level : 시스템 단위 로그 레벨을 입력한다. (DEBUG, INFO, WARN, ERROR, FATAL)

G. system.log.service.lever : 서비스 단위 로그 레벨을 입력한다. (DEBUG, INFO, WARN, ERROR, FATAL)

H. system.log.keep.date : 기록된 로그 파일을 보관할 일 수를 입력한다.

I. system.store.path : 실제 데이터 파일이 저장될 위치를 입력한다. 입력이 안되면 디폴트로 홈디렉토리의 /store 디렉토리 밑에 생성된다.

2) $SERVER_HOME/setting/{Service}/{Store}/store.properties

가) 서비스와 스토어 정보를 관리한다.

3) $SERVER_HOME/setting/$service_name/$store_name/$store_name.properties

가) 스토어의 속성 정보를 관리 (자세한 내용은 관리도구의 Chapter4 3 장의 스토어 메뉴 중 스토어 설정 부분을 참고한다.)

3. Server Work Process

3.1 서비스의 생성 및 삭제

1) 서비스의 생성

가) 서비스의 생성은 관리도구를 통해서 사용하는 방법을 추천하며 부득이하게 관리도구를 이용할 수 없다면 아래와 같이 시도한다.

나) 서비스의 생성은 $SERVER_HOME₩setting 디렉토리 안의 store.properties 파일을 수정하고 서버를 재시작 해야만 한다.

다) store.properties 파일의 관련 프로퍼티

A. store.service.name - 서버가 현재 서비스하고 있는 모든 서비스 목록. 각 항목은 띄어쓰기로 구분되어 있다.

B. [$serviceName].prefix – 쿼리에서 사용되는 그래프명에 대한 서비스의 프리픽스 경로 값으로 URI 형태로 설정해야 한다. (실제 그래프명은 프리픽스 경로 값+서비스명으로 자동 생성된다. 쿼리에서도 이 그래프 명을 이용해서 쿼리를 작성할 수 있다.)

C. [$serviceName].store.list - [$serviceName]가 서비스하고 있는 모든 스토어 목록. 각 항목은 띄어쓰기로 구분되어 있다.

D. [$serviceName].store.default - [$serviceName]의 디폴트 store 의 이름. Default store 는 서비스당 하나만 존재하여야 한다.

E. [$serviceName].description- [$serviceName]의 description 정보

라) 서비스 생성 방법

A. OntoBase2.0 서버를 정지한다. (정지하지 않고 생성하기 위해서는 반드시 관리도구를 통해서만 생성해야 한다.)

B. Store.service.name 항목에 새로운 항목의 서비스 이름을 추가한다. 기존에 서비스가 존재한다면 띄어쓰기로 구분한다.(서비스 명은 띄어쓰기를 지원하지 않는다.) 여기서는 sampleService 라고 지칭해 본다.

C. Store.service.name 항목에 sampleService 라고 입력 후 다음 항목을 수정(없다면 추가) 한다.

① [sampleService].store.list 항목에 서비스가 사용할 store 명을 입력한다. 여기서는 sampleStore 라고 기입한다. ([sampleService].store.list=sampleStore)

② [sampleService].store.default 항목에는 반드시 하나의 디폴트 스토어가 입력되어야 한다. 여기서는 sampleStore 라는 스토어를 하나 생성하였으므로 이 스토어 명을 입력한다. ([sampleService].store.default=sampleStore))

③ [sampleService].description 항목에는 해당 서비스에 대한 Description 내용을 기입한다.

D. 이 단계까지 항목을 추가하게 되면 기본적으로 새로운 서비스를 하나 생성하게 된다. 이제 이를 적용하기 위해서는 OntoBase2.0 서버를 재시작하거나 관리도구의 Server 메뉴를 통해 해당 서비스만을 시작할 수 있다.

2) 서비스의 삭제

가) 서비스는 여러 개의 Store 를 가지므로 삭제 시 주의를 요한다.

나) 서비스 삭제는 관리도구를 통해서 사용하는 방법을 추천하며 부득이하게 관리도구를 이용할 수 없다면 아래와 같이 시도한다.

다) 삭제 시나리오

A. 먼저 OntoBase2.0 서버를 반드시 정지한다. (정지하지 않고 삭제하기 위해서는 관리도구를 통해 삭제한다.)

B. $SERVER_HOME₩setting 디렉토리 안에 store.properties 파일을 연다

C. 만약 sampleService 라는 서비스를 삭제하고 싶다면 store.service.name 항목에서 sampleService 부분만을 삭제한다. (다른 서비스명이 있다면 다른 부분은 삭제하지 말 것.)

D. [sampleService]로 시작되는 store.list, store.default, description 부분을 같이 삭제한다.

E. OntoBase2.0 서버를 기동하면 해당 서비스가 삭제가 적용이 된다.

3.2 스토어의 생성 및 삭제

1) 스토어의 생성

가) 스토어를 생성하는 방법 역시 서비스를 생성하는 방법과 비슷하다. 이 경우에도 관리도구를 통한 방법을 적극 추천한다.

나) $SERVER_HOME₩setting 디렉토리 안에 store.properties 파일을 이용하여 수동으로 스토어를 생성한다.

다) 생성 시나리오

A. OntoBase2.0 서버를 정지한다.

B. 생성하고자 하는 Store 가 위치할 서비스 명을 먼저 알고 있어야 한다. (여기서는 sampleService 라고 가정하고 추가할 스토어 명은 addStore 라고 가정한다.)

C. [sampleService].store.list 항목에 새로운 스토어 명 addStore 를 입력한다. 기존의 스토어와 구분하기 위해서는 띄어쓰기를 해야 한다.

D. [sampleService].store.default 항목은 해당 서비스의 디폴트 스토어를 설정한다. 만약 addStore 가 sampleService 의 디폴트 스토어이기를 원한다면 기존의 스토어 명을 지우고 addStore 를 입력한다.

E. OntoBase2.0 서버를 재시작하면 새로운 스토어(addStore)가 추가된 것이 적용되어 서비스되게 된다.

라) 관리도구를 통한 스토어 생성방법은 Chapter4 의 3. OntoBase2.0 Manager 화면 구성을 참고한다

2) 스토어의 삭제

가) 스토어 생성과 마찬가지로 마찬 가지로 관리도구를 통한 삭제를 추천하며 수동으로 삭제할 경우 $SERVER_HOME₩setting 디렉토리 안에 store.properties 파일을 이용한다.

나) 삭제 시나리오

A. OntoBase2.0 서버를 정지한다.

B. 삭제하려는 스토어 (여기서는 addStore 라고 가정한다.) 를 [sampleService].store.list 항목에서 삭제한다.

C. 만약 삭제하려는 스토어(addStore)가 default 스토어라면 삭제할 수 없다. 이럴 경우에는 서비스 전체를 삭제하여야 하나 [sampleService].store.list 의 스토어

항목이 2 개 이상인 경우라면 다른 스토어로 default 스토어로 설정 후에 삭제가 가능하다.

D. OntoBase2.0 서버를 구동하면 스토어가 삭제되어 적용되어 서비스가 시작된다.

다) 관리도구를 통한 스토어 삭제 방법은 Chapter4 의 Manager 화면 구성을 참고한다

3.3 데이터 로딩 - 파일 빌드

파일 빌드는 트리플을 표현하는 언어로 작성된 파일들을 트리플 스토어에 넣는 빌드 프로세스를 말한다. 지원하는 파일은 RDF/XML, N3, NTriple, Turtle 형식의 파일을 지원한다.

1) File Full Build

가) 개요 - 일반적으로 초기에 스토어를 구성하는 경우에 트리플을 표현하는 언어로 작성된 파일들을 한꺼번에 트리플 스토어에 넣는 Bulk 로딩을 말한다. 이 빌드는 일반적인 빌드보다 고가용성의 로직을 통해 고속 빌드를 보장한다.

나) 실행

A. 스크립트로 실행 (빌드할 파일들이 서버에 존재하는 경우)

① $SERVER_HOME₩bin 디렉토리의 registFileBuild 파일을 수정해서 실행한다. (registFileBuild 파일에서 빌드할 파일이 존재하는 디렉토리를 셋팅하고 실행 타입을 Full Build 모드로 셋팅해야 한다.)

B. 클라이언트 라이브러리로 실행 (빌드할 파일들이 서버에 존재하는 경우)

① 클라이언트 라이브러리에 포함된 StoreManager.java 의 registFileFullBuild 메서드를 실행한다. (자세한 실행 방법은 Chapter5 의 주요 클라이언트 라이브러리 부분을 참조한다.)

2) File Increment Build

가) 개요 - 서버의 중지 없이 기존에 존재하는 스토어에 트리플을 표현하는 언어로 작성된 일정량의 파일들을 스토어에 넣는 빌드를 말한다. 이 빌드는 추가 빌드와 삭제 빌드를 구분하여 로딩이 이루어지며 데이터의 안정성을 보장한다.

나) 실행

A. 스크립트로 실행 (빌드할 파일들이 서버에 존재하는 경우)

① $SERVER_HOME₩bin 디렉토리의 registFileBuild 파일을 수정해서 실행한다. (registFileBuild 파일에서 빌드할 파일이 존재하는 디렉토리를 셋팅하고 실행 타입을 Increment Build 모드로 셋팅해야 한다.)

B. 클라이언트 라이브러리로 실행 (빌드할 파일들이 서버에 존재하는 경우)

① 클라이언트 라이브러리에 포함된 StoreManager.java 의 registFileAddIncrementBuild 나 registFileDeleteIncrementBuild 메서드를 실행한다. (자세한 실행 방법은 Chapter5 의 주요 클라이언트 라이브러리 부분을 참조한다.)

C. 클라이언트 라이브러리로 실행 (빌드할 파일들이 클라이언트에 존재하는 경우)

① 클라이언트 라이브러리에 포함된 StoreManager.java 의 addFiles 나 addFile 메서드를 실행한다. (자세한 실행 방법은 Chapter5 의 주요 클라이언트 라이브러리 부분을 참조한다.)

3.4 데이터 로딩 - 수동 빌드

수동 빌드는 임의의 트리플 셋을 수동으로 스토어에 추가할 수 있는 빌드 프로세스를 말한다.

1) Manual Add Build

가) 개요 - 서버가 가동 중인 경우에 임의의 트리플 셋을 스토어에 추가할 수 있는 빌드를 말한다.

나) 실행

A. 관리도구로 실행

① 관리도구의 Store > Work 에서 Manual Build 를 실행한다. (자세한 실행 방법은 Chapter4 의 Work 부분을 참조)

B. 클라이언트 라이브러리로 실행

① 클라이언트 라이브러리에 포함된 StoreManager.java 의 addModel 이나 addTriple 메서드를 실행한다. (자세한 실행 방법은 Chapter5 의 주요 클라이언트 라이브러리 부분을 참조한다.)

2) Manual Delete Build

✓ 개요 - 서버가 가동 중인 경우에 임의의 트리플 셋을 스토어에서 삭제할 수 있는 빌드를 말한다.

✓ 실행

A. 관리도구로 실행

① 관리도구의 Store > Work 에서 Manual Build 를 실행한다. (자세한 실행 방법은 Chapter4 의 Work 부분을 참조)

B. 클라이언트 라이브러리로 실행

① 클라이언트 라이브러리에 포함된 StoreManager.java 의 deleteModel 이나 deleteTriple 메서드를 실행한다. (자세한 실행 방법은 Chapter5 의 주요 클라이언트 라이브러리 부분을 참조한다.)

3.5 질의 처리 - Select 쿼리

1) SPARQL 의 Select 구문에 대한 처리로서 클라이언트는 라이브러리에 포함된 SparqlQuery.java 의 execute 메서드를 통해서 스트링 형태의 Select 질의를 서버에 요청하면 서버는 쿼리에 대한 결과를 SPARQLResult 에 담아 리턴한다.
2) 클라이언트는 SPARQLResult 로부터 ResultSet 형태의 결과를 가져올 수 있다. (자세한 실행 방법은 Chapter5 의 클라이언트 라이브러리 부분을 참조한다.)

3.6 질의 처리 - Ask 쿼리

1) SPARQL 의 Ask 구문에 대한 처리로서 클라이언트는 라이브러리에 포함된 SparqlQuery.java 의 execute 메서드를 통해서 스트링 형태의 Ask 질의를 서버에 요청하면 서버는 쿼리에 대한 결과를 SPARQLResult 에 담아 리턴한다.
2) 클라이언트는 SPARQLResult 로부터 boolean 형태의 결과를 가져올 수 있다. (자세한 실행 방법은 Chapter5 의 클라이언트 라이브러리 부분을 참조한다.)

3.7 질의 처리 - Construct 쿼리

1) SPARQL 의 Construct 구문에 대한 처리로서 클라이언트는 라이브러리에 포함된 SparqlQuery.java 의 execute 메서드를 통해서 스트링 형태의 Construct 질의를 서버에 요청하면 서버는 쿼리에 대한 결과를 SPARQLResult 에 담아 리턴한다.
2) 클라이언트는 SPARQLResult 로부터 Model 형태의 결과를 가져올 수 있다. (자세한 실행 방법은 Chapter5 의 클라이언트 라이브러리 부분을 참조한다.)

3.8 질의 처리 - Describe 쿼리

1) SPARQL 의 Describe 구문에 대한 처리로서 클라이언트는 라이브러리에 포함된 SparqlQuery.java 의 execute 메서드를 통해서 스트링 형태의 Describe 질의를 서버에 요청하면 서버는 쿼리에 대한 결과를 SPARQLResult 에 담아 리턴한다.
2) 클라이언트는 SPARQLResult 로부터 Model 형태의 결과를 가져올 수 있다. (자세한 실행 방법은 Chapter5 의 클라이언트 라이브러리 부분을 참조한다.)

Chapter 4

OntoBase2.0 Manager

Chapter 4. OntoBase2.0 Manager

1. OntoBase2.0 Manager 개요

1.1 시스템 개요

OntoBase2.0 Manager 는 서버에 대한 관리 역할을 수행하는 관리자를 위한 웹 관리도구로서 서버의 다양한 기능을 손쉽게 실행 가능하도록 지원한다. OntoBase2.0 웹 관리도구는 별도의 커스터마이징 과정을 필요로 한다.

2. OntoBase2.0 Manager 구성

2.1 구성

1) OntoBase2.0 Manager 는 웹 서버나 J2EE 컨테이너에 설치되어 구동된다.
2) 자세한 구성이나 설치 방법은 Chapter2 를 참조한다.

3. OntoBase2.0 Manager 화면 구성

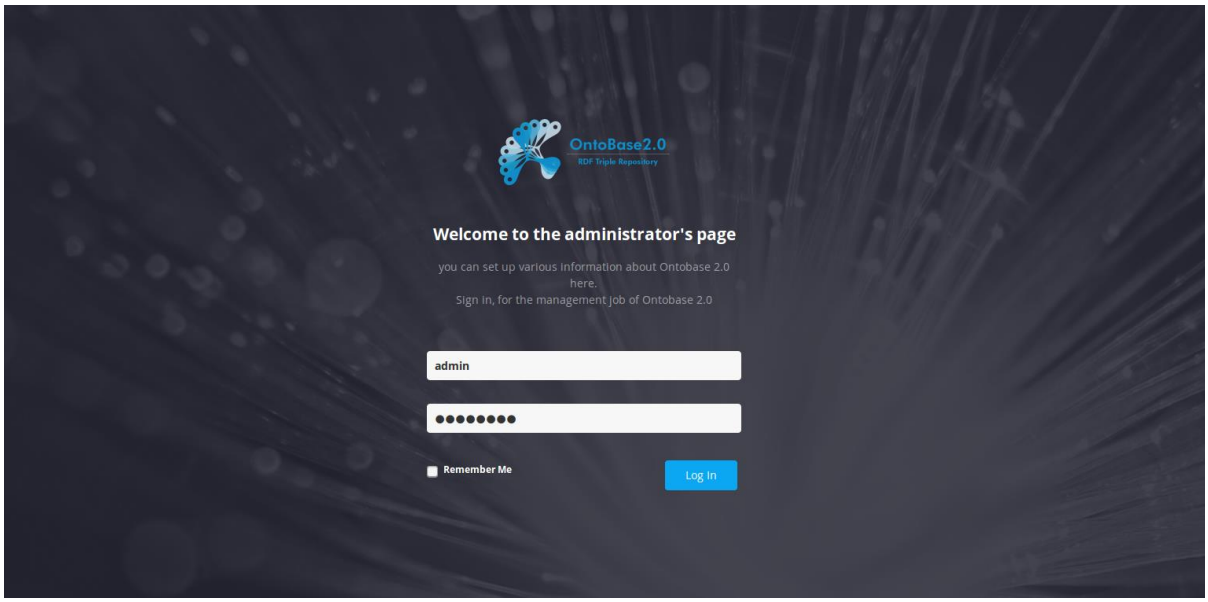
3.1 로그인

1) 초기 로그인 화면

A. 설치가 완료 되었다면 JSP 서블릿 엔진에 등록한 주소로 접속을 시도한다. (예: http://localhost:8080/mgr/index.onto)

B. 컨텍스트 등록이 성공하였다면 다음 화면에 접속할 수 있다.

C. 관리자용 아이디와 패스워드를 입력하면 초기 메인 화면으로 이동한다. (접속하기 위한 디폴트 관리도구 계정은 아이디는 admin, 패스워드는 admin@#$ 이다.)



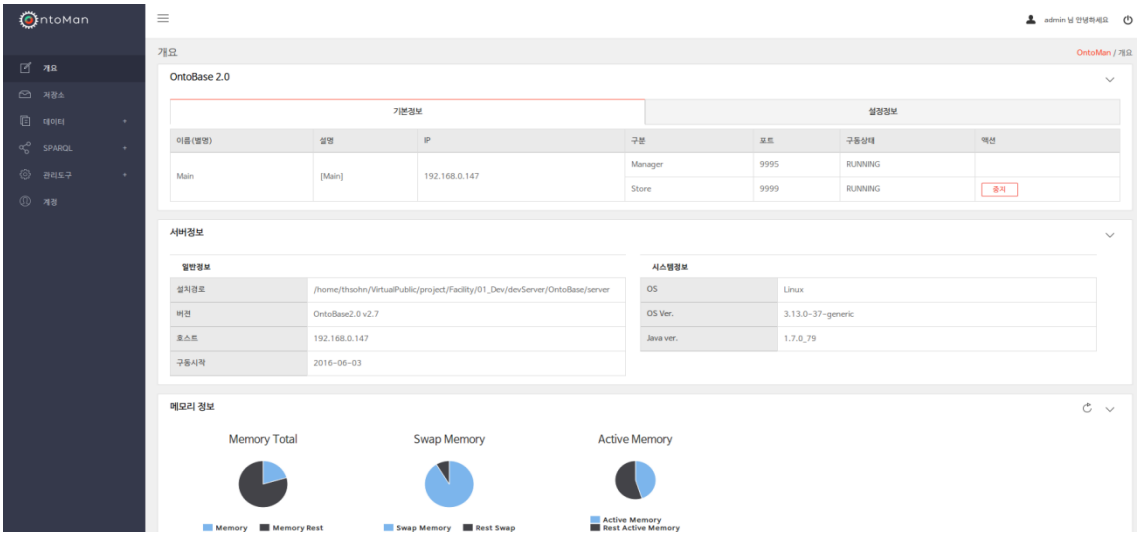
[4.1 로그인 화면]

2) 초기 접속 화면

A. 로그 인이 성공하면 다음의 초기 접속 화면인 개요 화면으로 이동한다.

B. 초기 접속 화면은 크게 좌측의 글로벌 메뉴바, 온투베이스 서버의 기본정보와 설정정보를 보여주는 탭과 서버 정보 및 메모리 정보로 구성되었다.

C. 우측 상단에는 환영 메시지와 함께 오른 쪽에는 로그아웃 버튼이 존재한다.



[그림 4.2 초기 접속 화면]

3.2 개요

1) 기본정보 탭과 설정정보 탭으로 구성되었다. 기본정보 탭에서는 온투베이스 서버의 이름, 위치 경로, 설치된 서버의 운영환경, 자바 버전, 운영서버의 메모리 정보 등 일반정보들을 개요형식으로 보여주며 설정정보 탭에서는 온투베이스 서버 정보를 변경하는 기능을 제공한다.

2) 기본정보

A. 현재 관리도구에 등록된 서버의 목록을 확인할 수 있다.

B. 이 화면에서는 서버의 정보와 서버의 동작을 관리한다.

C. [구분] 컬럼의 [Store] 서버가 정지된 상태에서는 해당 서버를 열람할 수 없다.

D. 화면 구성 항목

① 이름(별명) - 등록 서버의 이름.

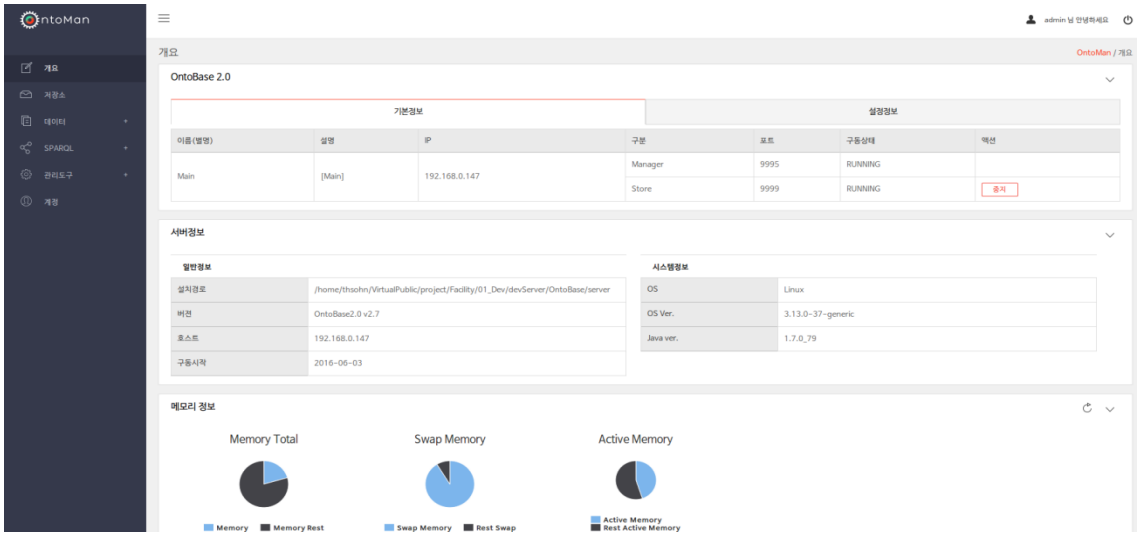
② 설명 - 서버 설명

③ IP - 서버가 바인딩 된 네트워크 주소

④ 포트 - 서버가 바인딩 된 포트 주소

⑤ 구동상태 - 서버의 동작 유무

⑥ 액션 - [시작], [중지] 버튼을 통해 서버의 시작과 종료를 실행한다.



[그림 4.3 기본정보 화면]

✓ 서버 시작시 초기화를 위한 약간의 지연시간이 존재합니다. 이와 같은 지연시간 때문에 서버의 상태가 stopped 로 표시될 수 도 있다. 저장소 서버 시작은 등록된 모든 스토어가 시작되어야만 관리도구에서 올바르게 관련 서버를 체크하므로, 서버를 완전히 구동된 것을 확인한 후 관리도구를 이용해야 한다.

3) 서버정보

A. 서버정보에는 서버환경의 개략정보를 보여주는 일반정보와 시스템 정보로 구성되었다.

B. 화면 구성 항목

① 설치경로 – 온투베이스 설치 경로

② 버전 – 온투베이스 버전 정보

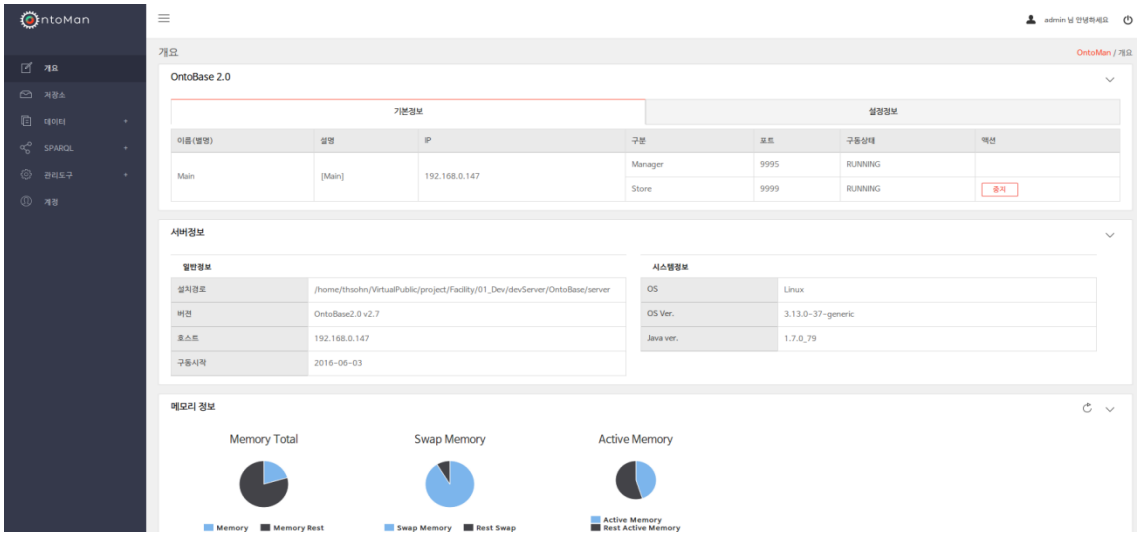
③ 호스트 – 온투베이스 설치된 서버의 호스트 주소

④ 구동시작 – 온투베이스 구동 시작일

⑤ OS - 온투베이스 설치된 서버의 운영체제

⑥ OS Ver. - 운영체제의 버전

⑦ Java ver. - 온투베이스에서 사용하고 있는 자바 버전



[그림 4.4 서버정보 화면]

4) 메모리 정보

A. 메모리 정보에는 토탈 메모리 정보와 스왑 메모리 및 액티브 메모리 정보로 구성되었으며 메모리 정보를 파이차트 형식으로 보여준다.

B. 화면 구성 항목

① Memory Total – 토탈 메모리

② Swap Memory - 스왑 메모리

③ Active Memory - 액티브 메모리

④ Garbage Collect - 메모리 정보 부분의 우측 상단의 버튼 클릭시 Garbage Collection 을 수행한다.

5) 설정정보

A. 설정정보에서는 서버 관련 프로퍼티 값들에 대한 설정을 변경할 수 있다.

B. 화면 구성 항목

① 서버 별칭 – 서버 별칭

② 현재스토어경로 - 온투베이스 스토어 경로

③ 시스템로그레벨 - 시스템 로그 레벨 설정

④ 쓰레드 풀 사이즈 - 쓰레드 풀 사이즈

⑤ 쓰레드 최대 사이즈 - 쓰레드 최대 사이즈

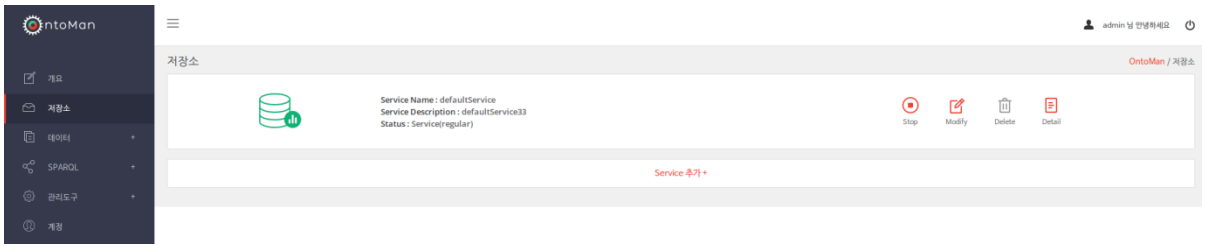
⑥ 이메일 공지 - 이메일 공지 관련 설정

3.3 저장소

1) 서비스 및 스토어의 추가, 수정, 삭제 등의 작업을 관리하고 각 스토어에 대한 정보들을 관리한다.
   A. 서비스를 추가, 수정, 삭제를 할 수 있다.
   B. 서비스 추가 - [Service 추가 +] 버튼을 클릭하면 서비스 추가 화면이 슬라이드 토글되어 저장소 메인 화면에 보여진다. 추가하려는 정보를 입력한 뒤 [확인] 버튼 클릭시 서비스가 추가된다.
   C. 서비스 수정 – [modify] 버튼을 클릭하면 수정 화면이 슬라이드 토글된다.
   D. 서비스 삭제 – [Delete] 버튼을 클릭하면 서비스를 삭제할 수 있다. 단 defaultService 는 삭제할 수 없다.
   E. 서비스 중지 - [Stop] 버튼을 클릭하면 서비스를 중지할 수 있다. 서비스가 중지되면 [Stop] 버튼은 [Start] 버튼으로 토글된다.
   F. 스토어 리스트 - 특정 서비스의 [Detail] 버튼을 클릭하면 서비스의 스토어정보를 리스트 형식으로 슬라이드 토글되어 보여진다.
   G. 스토어 수정 - 스토어 리스트가 보여지는 상태에서 , 특정 스토어의 [수정] 버튼을 클릭하면 스토어 정보가 수정된다.
   H. 스토어 중지 - 스토어 리스트가 보여지는 상태에서 , 특정 스토어의 [중지] 버튼을 클릭하면 서비스 중인 스토어가 중지되며 [중지] 버튼은 [시작] 버튼으로 토글된다.
   I. 스토어 삭제 - 스토어 리스트가 보여지는 상태에서 , 특정 스토어의 [삭제] 버튼을 클릭하면 스토어가 삭제된다. 단 , defaultStore 는 삭제할 수 없다.
   J. 스토어 추가 - 스토어 리스트가 보여지는 상태에서 스토어리스트 오른쪽 상단에 [추가+] 버튼을 클릭하면 스토어 추가 화면이 슬라이드 토글되어 화면에 보여진다. 추가할 스토어 정보를 입력하고 [확인] 버튼을 클릭하면 해당 스토어가 서비스에 추가된다.
   K. 화면 구성 항목
      ① Service Name - 서비스 이름
      ② Service Description - 서비스 설명
      ③ Status - 서비스의 동작 상태를 표시한다.

④ Operation - [modify], [delete] 버튼을 통해 서비스의 추가 및 삭제 동작을 관리한다.

⑤ [add] 버튼 - 서비스의 추가화면으로 이동한다.



[그림 4.5 저장소 Main 화면]

2) 서비스 추가

A. 새로운 서비스를 추가 할 수 있다.

B. 화면 구성 항목

① Service Name - 서비스 이름

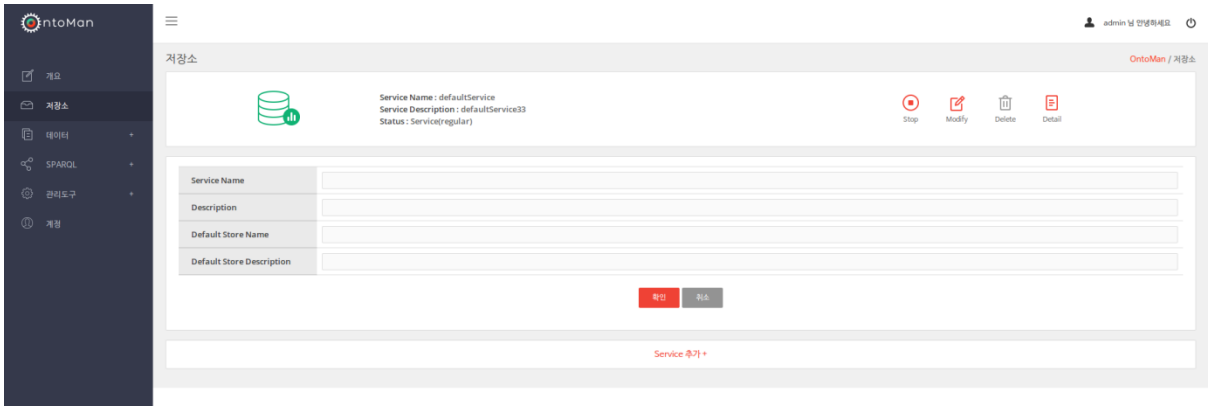
② Description - 서비스 설명

③ Default Store Name - 서비스가 가지는 디폴트 스토어 명

④ Default Store Description - 디폴트 스토어 설명

⑤ [확인] 버튼 - 서비스가 추가되고 저장소 Main 화면으로 이동한다.

⑥ [취소] 버튼 – 서비스의 추가를 취소하고 서비스 추가 화면은 슬라이드 토글되어 사라진다.



[그림 4.6 서비스 추가 화면]

3) 서비스 수정

A. 기존 서비스를 수정 할 수 있다.

B. 화면 구성 항목

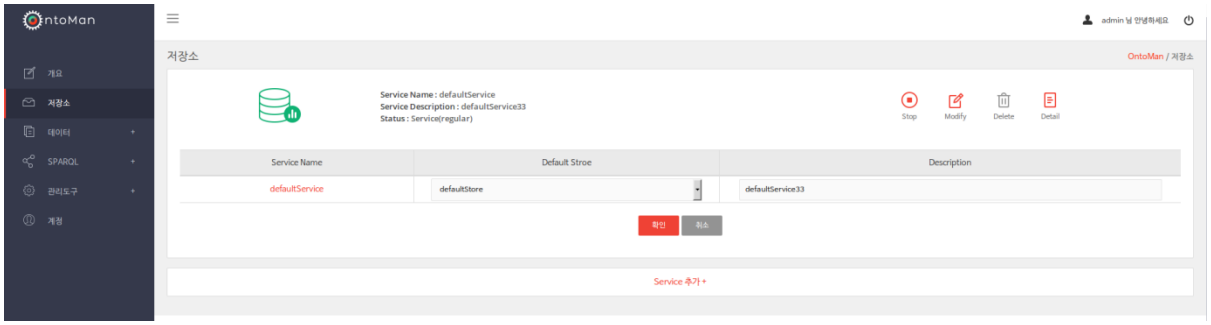
① Service Name - 서비스 이름으로 수정이 불가능하다.

② Description - 서비스 설명

③ Default Store - 서비스가 가지는 디폴트 스토어 명

④ [확인] 버튼 - 서비스의 수정을 실행한다.

⑤ [취소] 버튼 - 서비스의 수정을 취소하고 이전 화면으로 돌아간다.



[그림 4.7 서비스 수정 화면]

4) 스토어 추가

A. 새로운 스토어를 추가 할 수 있다.

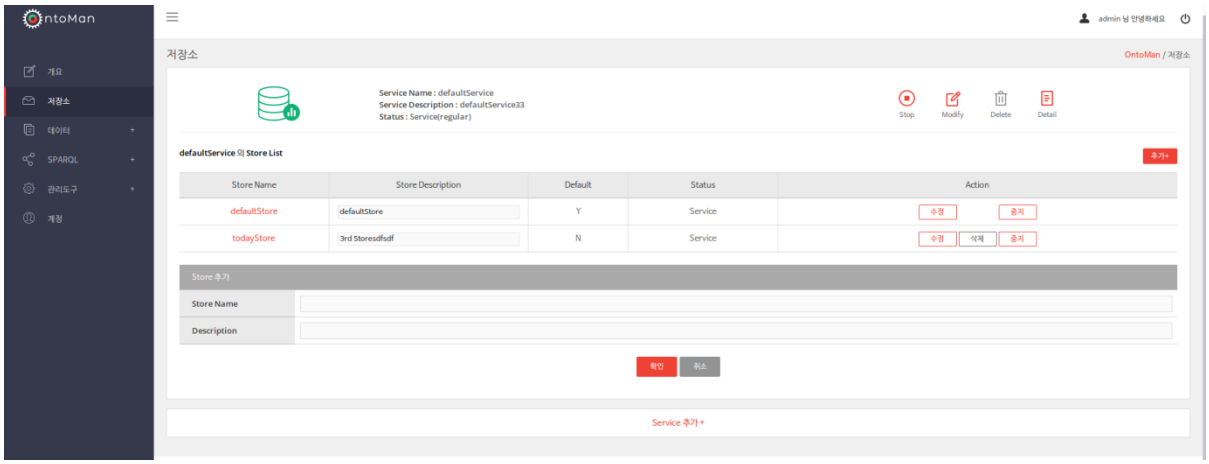
B. 화면 구성 항목

① Store Name - 스토어 명

② Description - 스토어 부가 설명

③ [확인] 버튼 - 스토어의 추가를 실행한다.

④ [취소] 버튼 - 스토어 추가를 취소하고 이전화면으로 돌아간다.



[그림 4.8 스토어 추가 화면]

5) 스토어 수정, 삭제 및 중지

A. 기존 스토어를 수정 할 수 있다.

B. 화면 구성 항목

① Store Name - 스토어 이름으로 수정이 불가능하다.

② Store Description - 스토어의 부가 설명

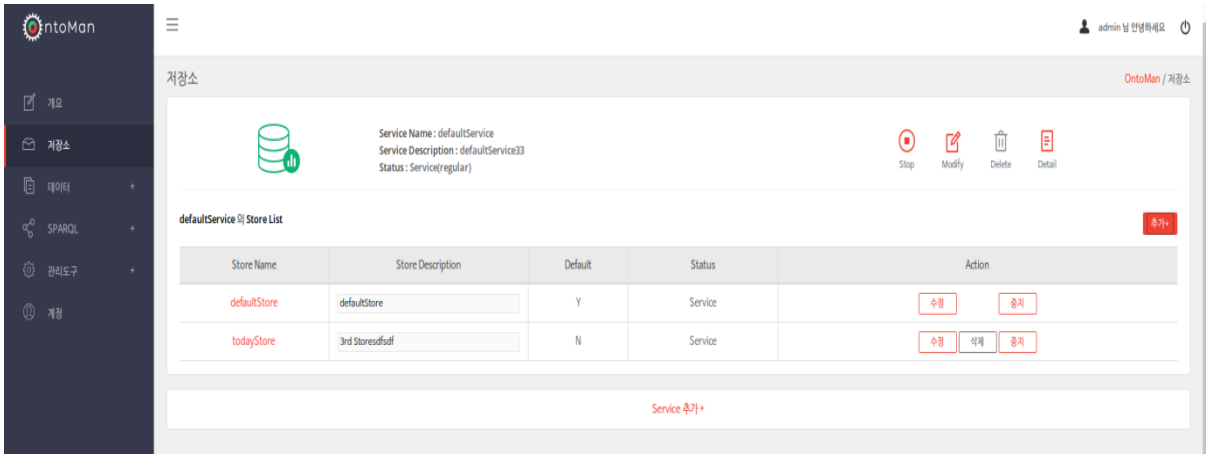
③ Default - 디폴트 스토어 여부로서 수정이 불가능하다.

④ Status - 스토어 구동 상태

⑤ [수정] 버튼 - 스토어의 수정을 실행한다.

⑥ [삭제] 버튼 - 스토어를 삭제한다.

⑦ [중지] 버튼 - 스토어를 중지한다.



[그림 4.9 스토어 수정,삭제 및 중지 화면]

3.4 데이터

1) 빌드

A. 특정 서비스의 스토어에 대하여 빌드를 수행할 수 있다. 수행할 수 있는 빌드 유형에는 전체빌드, 파일빌드, 수동빌드가 있다.

B. 전체빌드 : 전체빌드는 빌드할 디렉토리에 포함된 트리플데이터를 선택된 스토어에 전체빌드를 수행하는 것이며, 이 과정에서 스토어에 적재된 트리플 데이터는 전부 삭제되고 새로운 트리플들이 저장소에 적재된다.

C. 파일빌드 : 파일빌드는 하나의 파일을 특정 스토어에 빌드하는 것으로서, insert build, add build 와 delete build 가 있다. Insert build 는 저장소의 트리플들을 다 지우고 파일 내의 트리플 데이터를 저장소에 빌드하는 과정이고 add build 는 하나의 파일에 있는 트리플 데이터를 스토어에 추가하는 빌드과정이며 delete build 는 파일에 있는 트리플 데이터를 스토어에서 삭제하는 빌드유형이다.

D. 수동빌드 : 수동빌드는 사용자가 임의의 트리플을 수동으로 작성하여 저장소에 빌드하는 유형이다. 수행할 수 있는 타입으로는 저장소에 insert 할 수 있는 insert 타입과 삭제할 수 있는 delete 타입이 있다.

E. 빌드히스토리 : 선택된 저장소의 빌드 히스토리 목록을 조회한다.

F. 화면 구성 항목

① Select Service - 빌드작업을 수행할 대상 서비스를 선택한다.

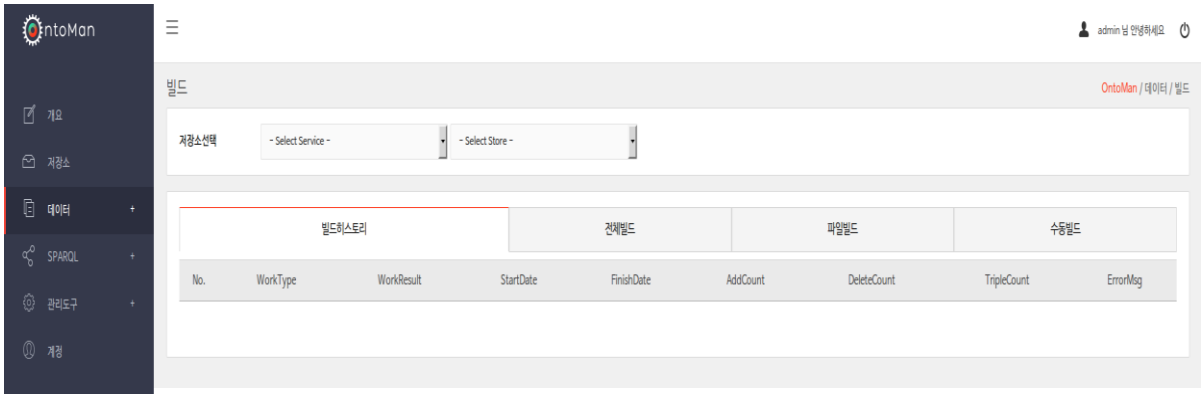
② Select Store - 빌드작업을 수행할 대상 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택된 대상 서비스에 종속된다.

③ 전체빌드 탭 : 전체빌드 작업을 수행하기 위한 기능 탭

④ 파일빌드 탭 : 파일빌드 작업을 수행하기 위한 기능 탭

⑤ 수동빌드 탭 : 수동빌드 작업을 수행하기 위한 기능 탭

⑥ 빌드히스토리 탭 : 빌드 히스토리를 조회하기 위한 기능 탭



[그림 4.10 데이터 - 빌드 메인 화면]

2) 빌드 - 전체빌드

A. 전체빌드는 빌드할 디렉토리에 포함된 트리플 데이터를 선택된 스토어에 전체빌드를 수행하는 것이다. 어에 전체빌드를 수행하는 것이다.

B. 화면 구성 항목

① Select Service - 빌드작업을 수행할 대상 서비스를 선택한다.

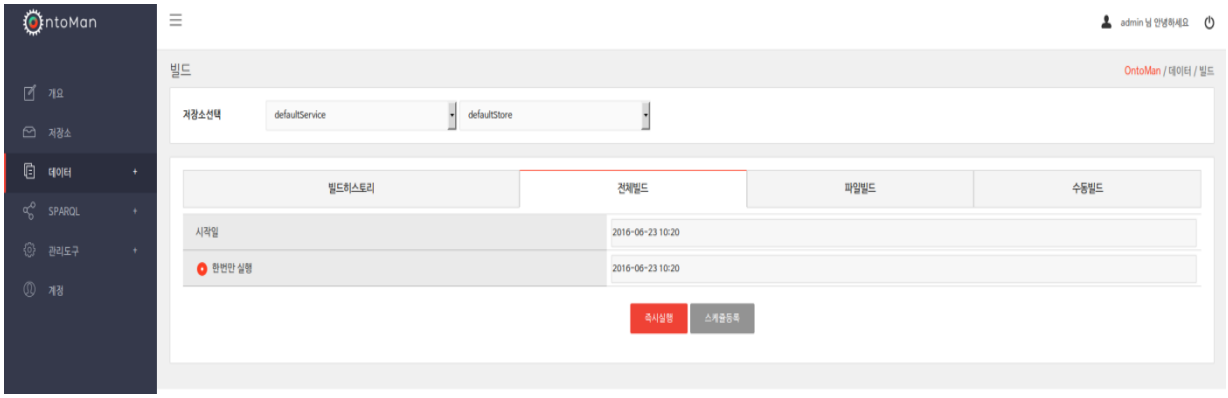
② Select Store - 빌드작업을 수행할 대상 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택된 대상 서비스에 종속된다.

③ 시작일 : 전체빌드를 수행할 시작일을 지정한다.

④ 한번만 실행 : 시작일을 기준으로 한번만 실행할 날자를 설정하여 스케쥴에 등록한다.

⑤ [즉시실행] 버튼 : 전체빌드를 즉시 실행한다.

⑥ [스케쥴등록] 버튼 : 사용자가 설정한 전체빌드 스케줄을 등록한다.



[그림 4.11 전체빌드 화면]

3) 빌드 - 파일빌드

A. 파일빌드는 하나의 파일을 특정 스토어에 빌드하는 것으로서, insert build, add build 와 delete build 가 있다.

B. 화면 구성 항목

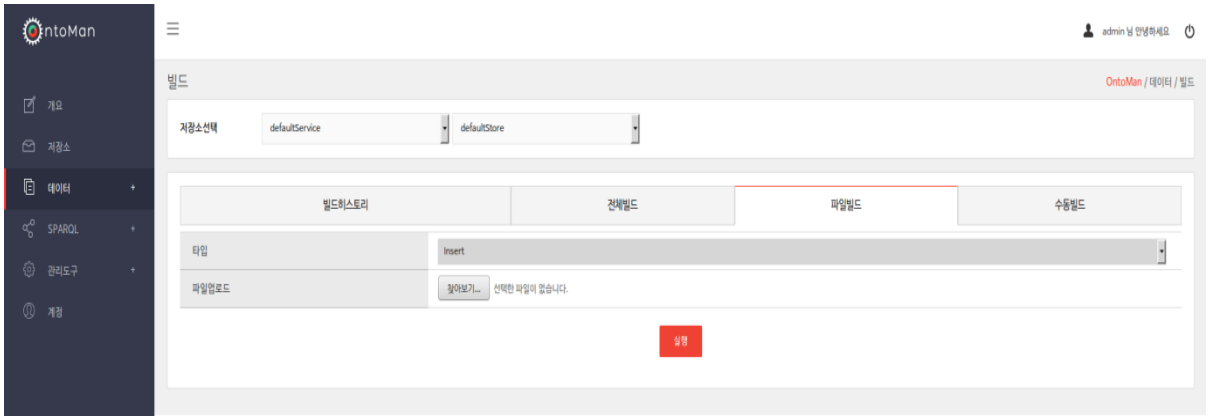
① Select Service - 빌드작업을 수행할 대상 서비스를 선택한다.

② Select Store - 빌드작업을 수행할 대상 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택된 대상 서비스에 종속된다.

③ 타입 : 빌드 타입을 선택할 수 있으며 선택할 수 있는 타입은 Insert , Add 와 Delete 이다.

④ 파일업로드 : 빌드할 파일을 선택한다.

⑤ [실행] 버튼 : 파일빌드를 수행한다.



[그림 4.12 파일빌드 화면]

4) 빌드 - 수동빌드

A. 수동빌드는 사용자가 임의의 트리플을 수동으로 작성하여 저장소에 빌드하는 작업이다.

B. 화면 구성 항목

① Select Service - 빌드작업을 수행할 대상 서비스를 선택한다.

② Select Store - 빌드작업을 수행할 대상 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택된 대상 서비스에 종속된다.

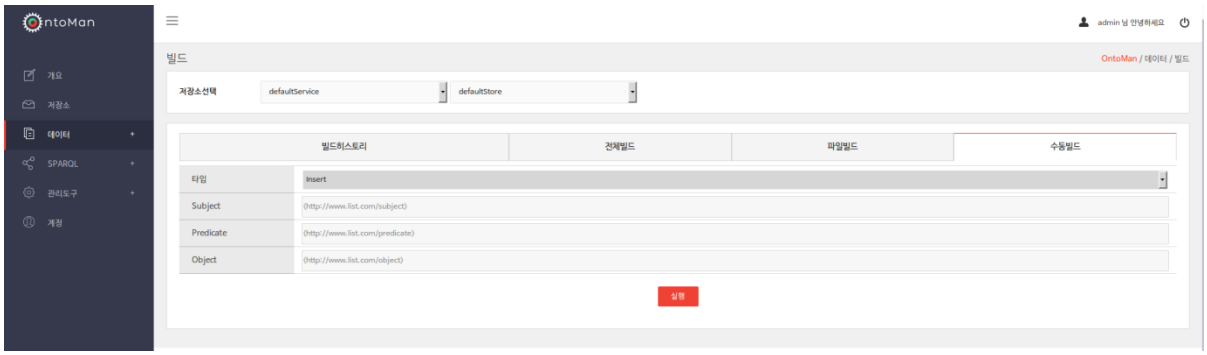
③ 타입 : 수동 빌드 타입을 선택할 수 있으며 선택할 수 있는 타입은 Insert 와 delete 가 있다.

④ Subject : 트리플의 Subject (주어) 부분을 작성한다. 리소스 형식으로 작성되어야 한다.

⑤ Predicate : 트리플의 Predicate (술어) 부분을 작성한다. 리소스 형식으로 작성되어야 한다.

⑥ Object : 트리플의 Object (목적어) 부분을 작성한다. 리소스 형식으로 작성되어야 한다.

⑦ [실행] 버튼 : 수동빌드를 수행한다.



[그림 4.13 수동빌드 화면]

5) 빌드 - 빌드히스토리

A. 선택된 저장소의 빌드 히스토리 목록을 조회할 수 있다.

B. 화면 구성 항목

① Select Service - 빌드작업을 수행할 대상 서비스를 선택한다.

② Select Store - 빌드작업을 수행할 대상 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택된 대상 서비스에 종속된다.

③ No - 일련번호

④ WorkType - 빌드 타입

⑤ WorkResult - 빌드 수행 결과 성공여부

⑥ StartDate - 빌드 수행 시작 시간

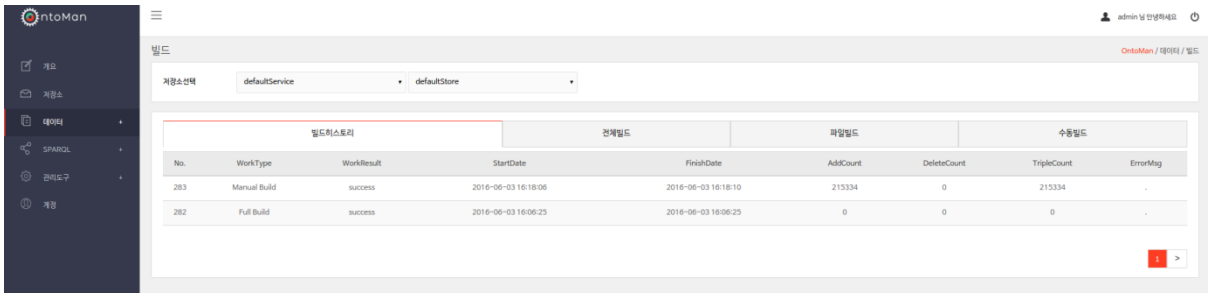
⑦ FinishDate - 빌드 수행 마감 시간

⑧ AddCount - 추가된 트리플 갯수

⑨ DeleteCount - 삭제된 트리플 갯수

⑩ TripleCount - 총 트리플 갯수

⑪ ErrorMsg - 에러가 발생했을 경우 에러 메시지



[그림 4.14 빌드히스토리 화면]

6) 스키마

A. 관리자가 선택한 저장소의 스키마 정보를 볼 수 있다.

B. Classes - 저장소의 클래스 목록을 조회한다.

C. Properties - 저장소의 프로퍼티 목록을 조회한다.

D. Synchronization - 트리플 스토어에 적재된 온톨로지 모델이 변경되었을 경우 관리도구에서 사용하고 있는 정보를 최신으로 유지하기 위하여 동기화 수행한다.

E. 화면 구성 항목

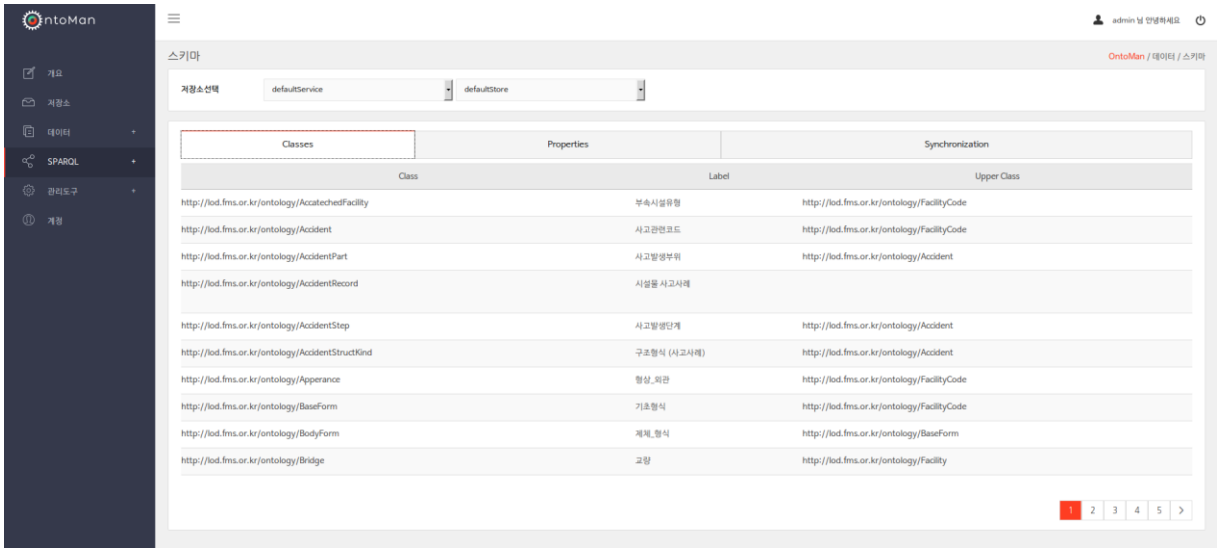
① Select Service - 스키마 조회 작업을 수행할 대상 서비스를 선택한다.

② Select Store - 스키마 조회 작업을 수행할 대상 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택한 대상 서비스에 종속된다.

③ Classes 탭 - 특정 저장소의 클래스 정보를 조회한다.

④ Properties 탭 - 특정 저장소의 프로퍼티 정보를 조회한다.

⑤ Synchronization 탭 - 동기화 기능을 수행한다.



[그림 4.15 스키마 메인 화면]

7) 스키마 - 클래스

A. 관리자가 선택한 저장소의 클래스 정보를 조회한다. 클래스 정보에는 클래스 URI, Label 및 상위 클래스 URI 정보로 구성된다.

B. 화면 구성 항목

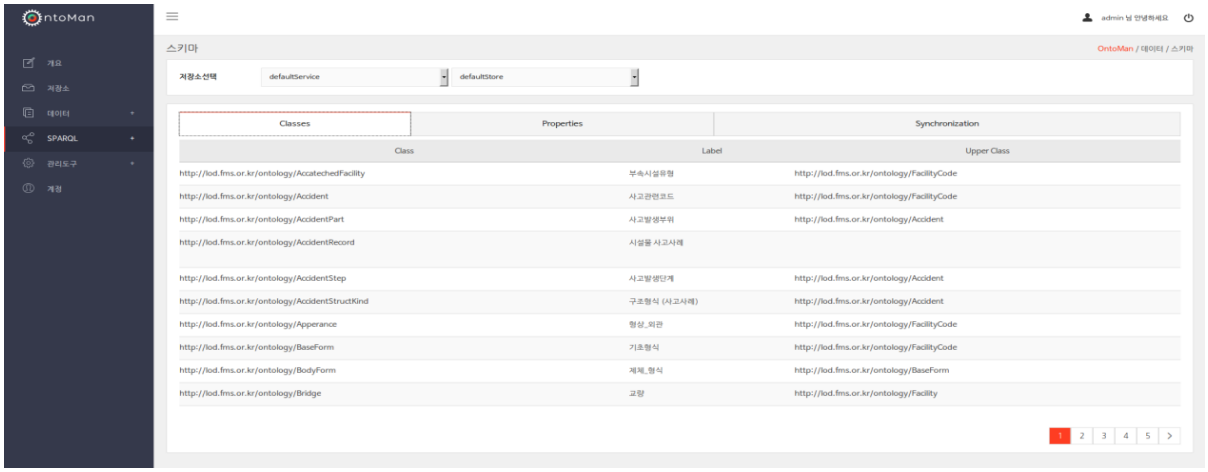
① Select Service - 스키마 조회 작업을 수행할 대상 서비스를 선택한다.

② Select Store - 스키마 조회 작업을 수행할 대상 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택한 대상 서비스에 종속된다.

③ Class - 클래스의 URI 정보

④ Label - 클래스의 라벨 정보

⑤ Upper Class - 상위 클래스의 URI 정보



[그림 4.16 스키마 - 클래스 화면]

8) 스키마 - 프로퍼티

A. 관리자가 선택한 저장소의 프로퍼티 정보를 조회한다. 프로퍼티 정보에는 프로퍼티 URI, Label, 프로퍼티 구분(Datatype & Object), 및 도메인, 레인지 정보로 구성된다.

B. 화면 구성 항목

① Select Service - 스키마 조회 작업을 수행할 대상 서비스를 선택한다.

② Select Store - 스키마 조회 작업을 수행할 대상 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택한 대상 서비스에 종속된다.

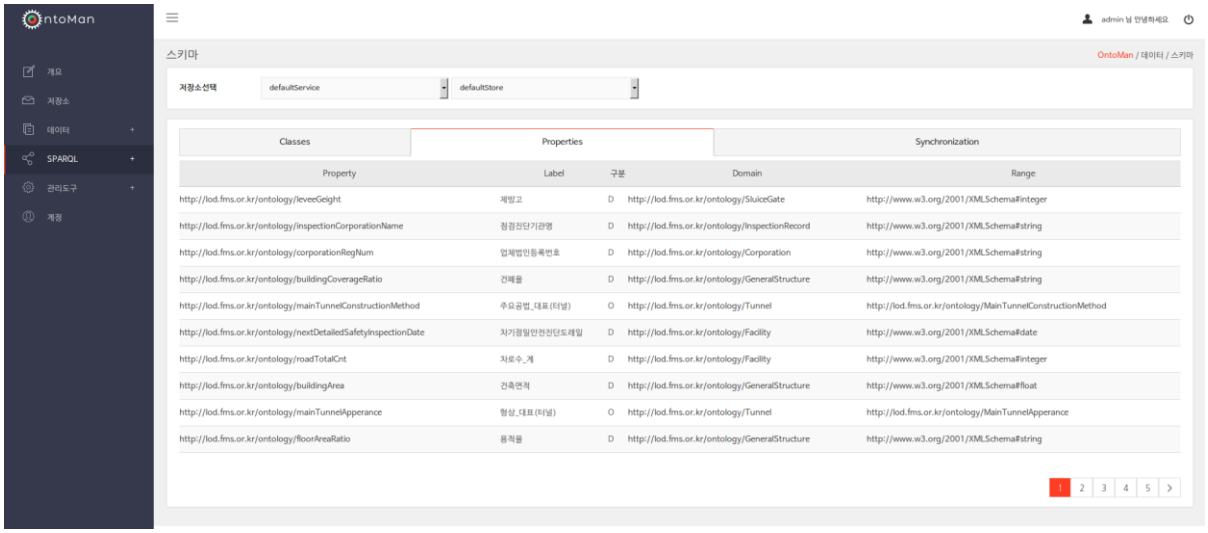
③ Property - 프로퍼티 URI

④ Label - 프로퍼티 레이블

⑤ 구분 - DatatypeProperty 와 ObjectProperty 에 대한 구분

⑥ Domain - 프로퍼티의 도메인 리소스의 URI

⑦ Range - 프로퍼티의 레인지 노드의 URI



[그림 4.17 스키마-프로퍼티 화면]

9) 스키마 - 동기화

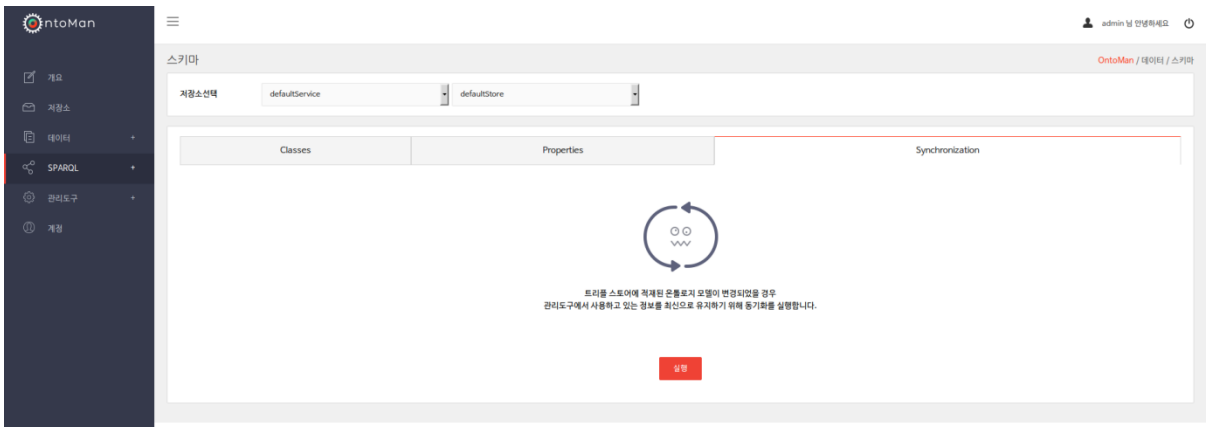
A. 트리플 스토어에 적재된 온톨로지 모델이 변경되었을 경우 관리도구에서 사용하고 있는 정보를 최신으로 유지하기 위해 동기화를 실행한다.

B. 화면 구성 항목

① Select Service - 스키마 조회 작업을 수행할 대상 서비스를 선택한다.

② Select Store - 스키마 조회 작업을 수행할 대상 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택한 대상 서비스에 종속된다.

③ [실행] 버튼 - 버튼 클릭시 동기화를 수행한다.



[그림 4.18 스키마-동기화 화면]

10) 변환

A. 관리자가 작성한 변환규칙을 이용하여 대상 데이터를 수집하고 트리플 변환을 수행한다.

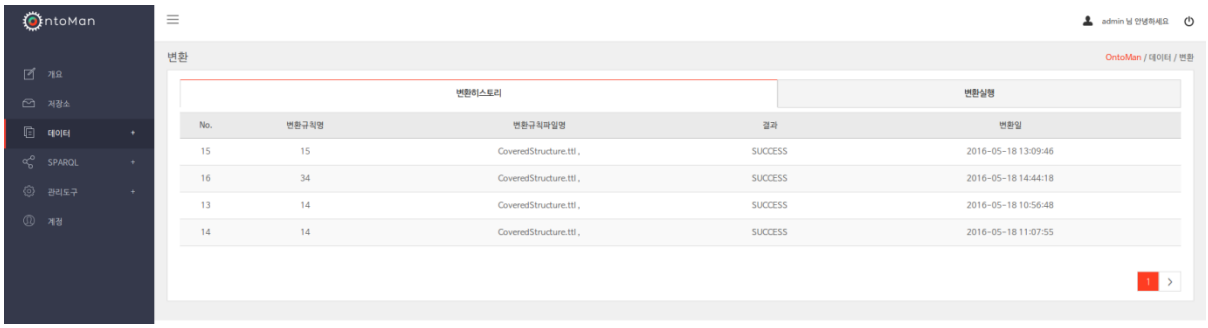
B. 변환실행 - 트리플 변환 규칙 파일을 실행하는 기능을 수행한다.

C. 변환히스토리 - 변환 이력을 조회한다.

D. 화면 구성 항목

① 변환히스토리 탭 - 변환 이력을 조회한다.

② 변환실행 탭 - 변환실행 기능을 수행한다.



[그림 4.19 변환 메인 화면]

11) 변환 - 변환히스토리

A. 변환 수행 히스토리 목록을 조회한다.

C. 화면 구성 항목

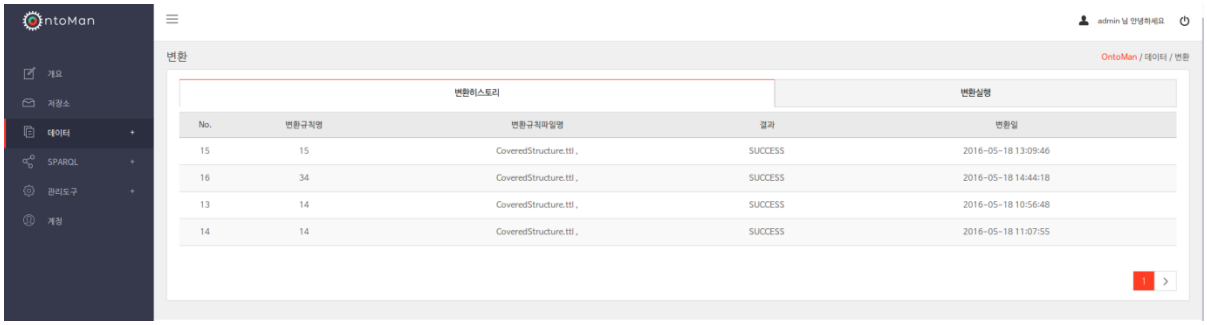
① No. - 일련번호

② 변환규칙명 - 변환규칙 이름

③ 변환규칙 파일명 - 관리자가 작성한 변환규칙 파일명

④ 결과 - 변환 수행 결과

⑤ 변환일 - 변환 수행 시간



[그림 4.20 변환히스토리 화면]

12) 변환 - 변환실행

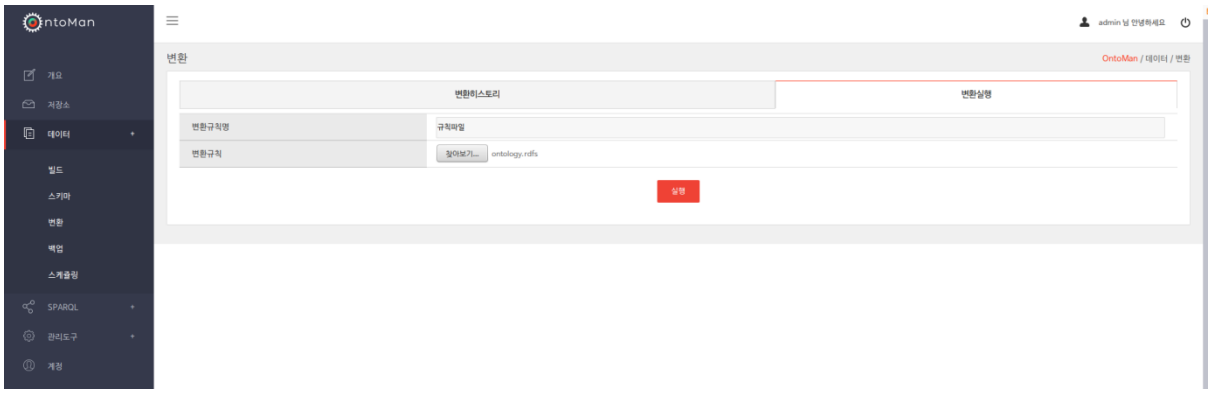
A. 변환규칙을 통하여 대상 데이터 수집 및 변환을 수행하여 트리플 데이터를 적재한다.

B. 화면 구성 항목

① 변환규칙명 - 변환규칙 이름

② 변환규칙 - 작성된 변환규칙 파일을 선택한다.

③ [실행] 버튼 - 변환을 수행한다.



[그림 4.21 변환실행 화면]

13) 백업

A. 스토어 단위로 백업작업을 수행한다.

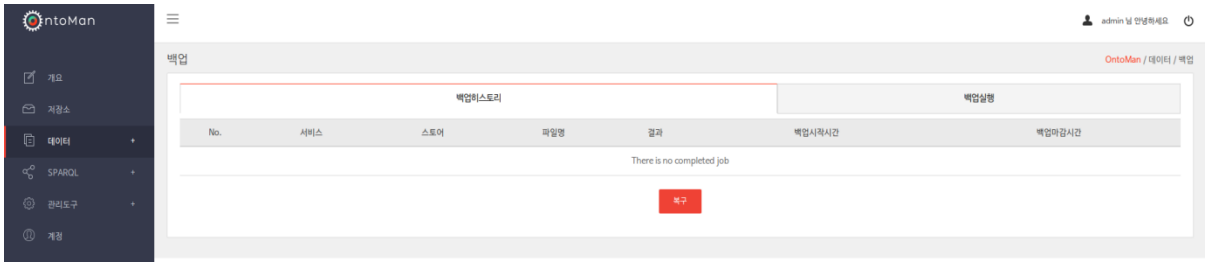
B. 백업히스토리 - 백업 이력을 조회한다.

C. 백업실행 - 백업기능을 수행한다.

D. 화면 구성 황목

① 백업히스토리 탭 - 백업 이력을 조회한다.

③ 백업실행 탭 - 백업실행 기능을 수행한다.



[그림 4.22 백업 메인 화면]

14) 백업 - 백업히스토리

A. 백업 수행 히스토리 목록을 조회한다.

B. 화면 구성 항목

① No. - 일련번호

② 서비스 - 백업할 대상 스토어 서비스 이름

③ 스토어 - 백업할 대상 스토어 이름

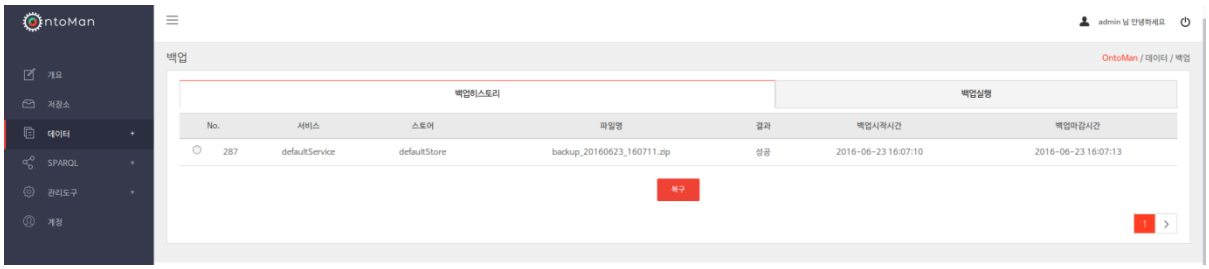
④ 파일명 - 백업파일 이름

⑤ 결과 - 백업 작업 수행 결과

⑥ 백업시작시간 - 백업 시작 시간

⑦ 백업마감시간 - 백업 마감 시간

⑧ [복구] 버튼 - 백업한 스토어 복구 기능 수행



[그림 4.23 백업 히스토리 화면]

15) 백업 - 백업실행

A. 백업할 저장소를 선택하여 백업 작업을 수행한다.

B. 화면 구성 항목

① 백업대상 - 백업할 대상 서비스와 스토어를 선택한다.

② 한번만 실행 - 백업을 한번만 수행할 경우, 실행 시간을 선택한다.

③ 주기적 실행 - 백업을 주기적으로 수행할 경우, 백업 실행 주기를 일, 시간, 분 단위로 입력한다.

④ [즉시실행] 버튼 - 백업작업을 즉시 실행한다.

⑤ [스케쥴등록] 버튼 - 한번만 실행 혹은 주기적 실행 선택한 뒤 스케쥴등록을 실행한다.

[그림 4.24 백업실행 화면]

16) 스케쥴링

A. 등록된 스케쥴 목록을 조회한다.

B. 화면 구성 항목

① No. - 일련번호

② 구분 - 작업 구분

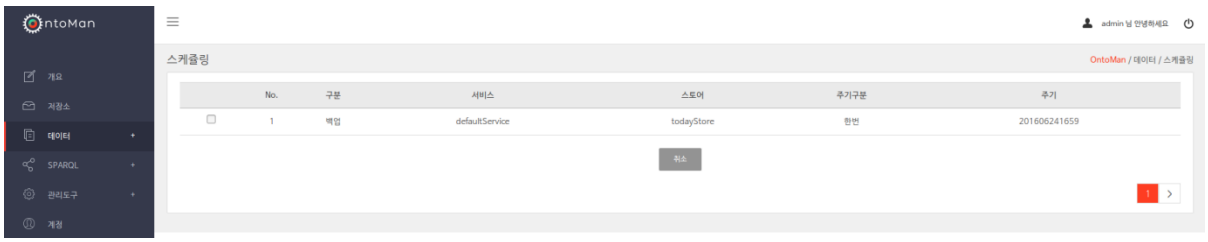
③ 서비스 - 스케쥴 대상 서비스

④ 스토어 - 스케쥴 대상 스토어

⑤ 주기구분 - 스케쥴 주기 구분 (주기적 / 일회성)

⑥ 주기 - 스케쥴 주기

⑦ [취소] 버튼 - 스케쥴 취소 기능을 수행한다.



[그림 4.25 스케쥴링 화면]

3.5 SPARQL

1) 쿼리 테스트

A. 서비스가 동작하고 있다면 해당 페이지를 통해 간단한 질의를 서버로 전송하고 결과를 확인할 수 있다.

B. 화면 구성 항목

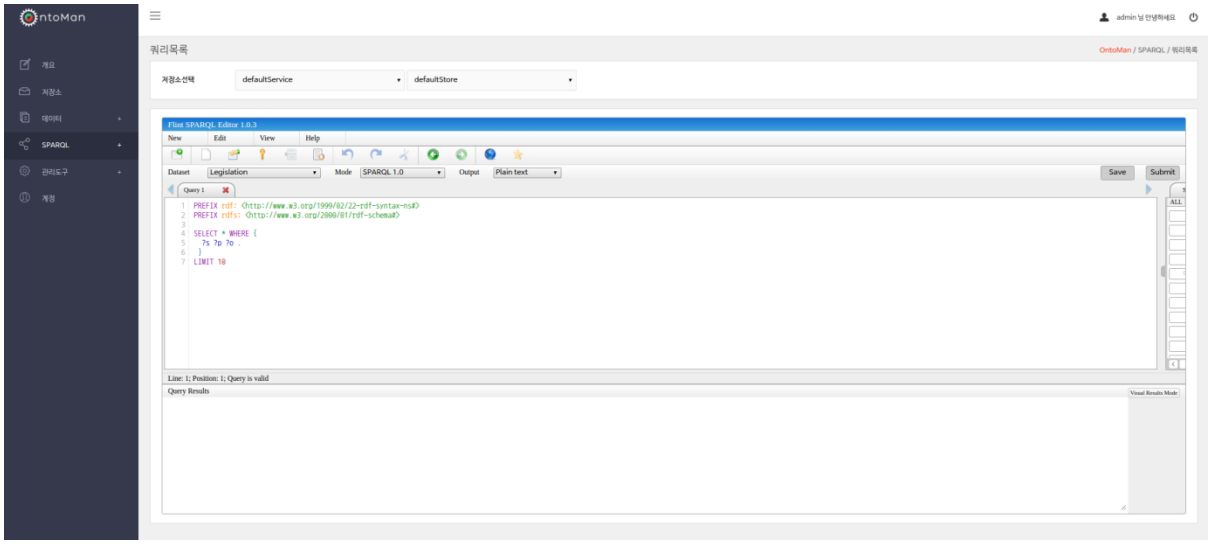
① Select Service - 쿼리 테스트를 수행할 대상 서비스를 선택한다.

② Select Store - 쿼리 테스트를 수행할 대상 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택된 대상 서비스에 종속된다.

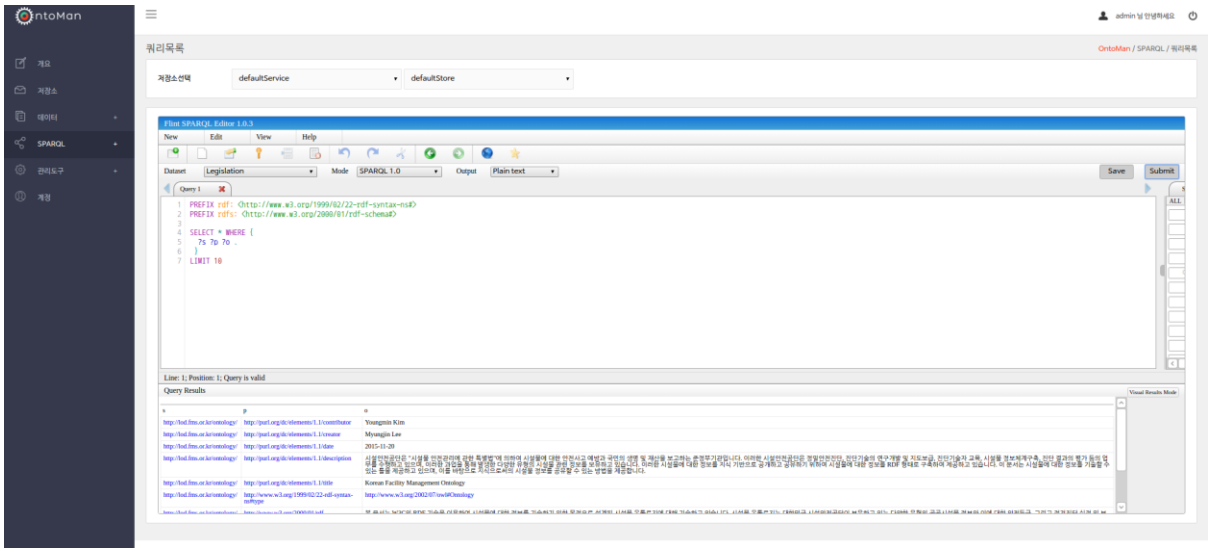
③ 쿼리의 오른쪽 버튼들 - SPARQL 기본 예약어에 대한 로직을 TextBox 에 자동으로 입력한다. (SPARQL 에서 지원하는 다양한 예약어를 화면에 보여주고 클릭하면 자동으로 예약어 에해당하는 구문을 입력해준다.)

④ [Submit] 버튼 - 쿼리 질의를 실행한다. 질의에 대한 결과는 아래에 보여준다.

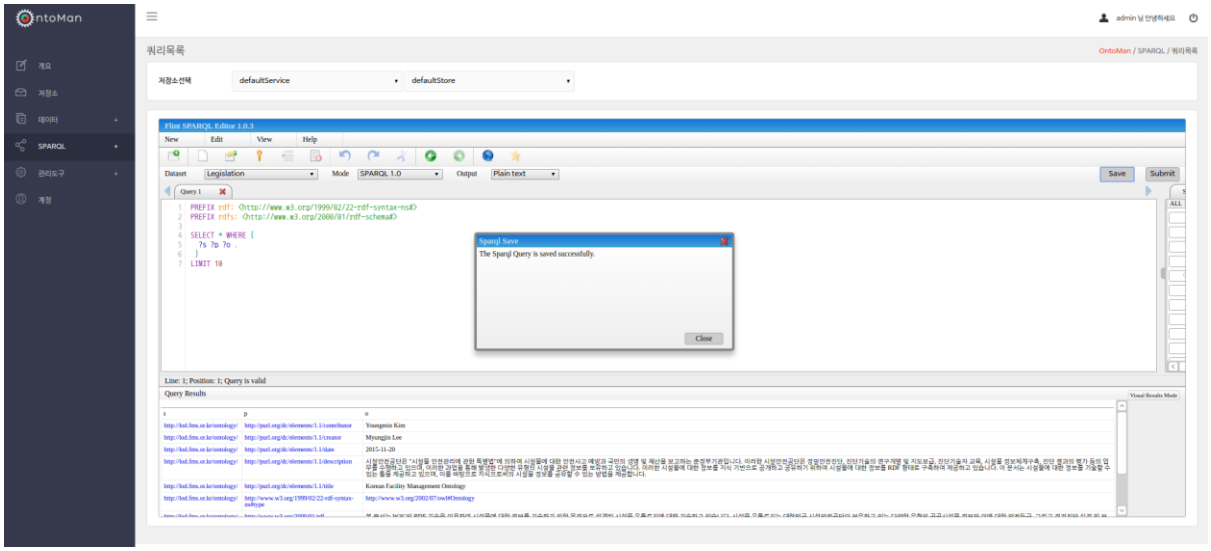
⑤ [Save] 버튼 - 쿼리를 저장한다.



[그림 4.26 쿼리 테스트 화면]



[그림 4.27 쿼리 테스트 결과화면]



[그림 4.28 쿼리 저장 화면]

2) 쿼리 목록

A. 쿼리 테스트에서 쿼리를 작성하고 저장한 SPARQL 쿼리 목록을 보여준다.

B. 화면 구성 항목

① 체크박스 - 쿼리를 선택하기 위한 체크박스

② No. - 일련번호

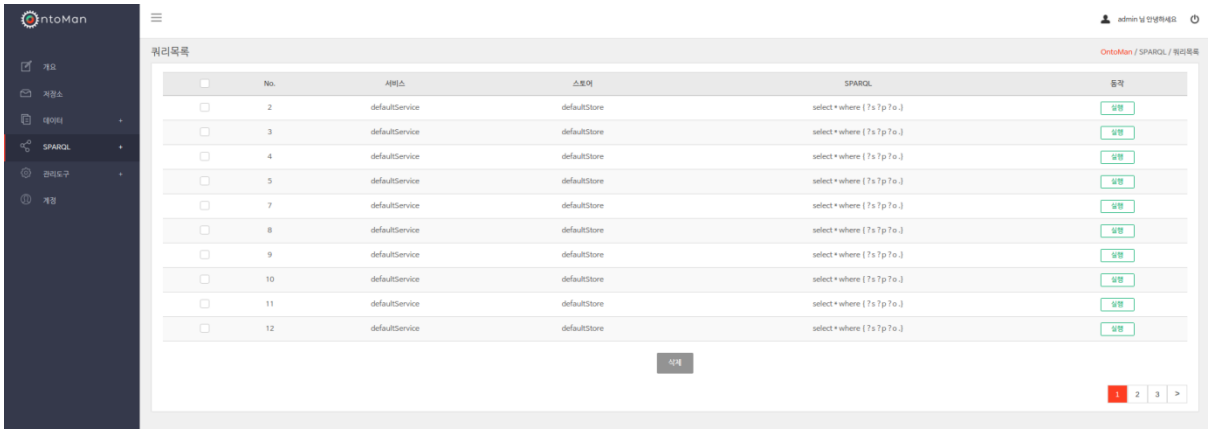
③ 서비스 - 쿼리 질의를 수행할 대상 서비스

④ 스토어 - 쿼리 질의를 수행할 대상 스토어

⑤ SPARQL - SPARQL 질의 쿼리

⑥ [실행]버튼 - SPARQL 쿼리를 쿼리테스트의 쿼리 TextBox 에 입력하여 실행할 수 있다.

⑦ [삭제]버튼 - 선택된 SPARQL 쿼리를 삭제한다.



[4.29 쿼리 목록 화면]

3.6 관리도구

1) 추론

A. 추론은 추론의 레벨을 정하기 위한 설정 및 현재 시스템에서 사용하고 있는 추론규칙에 대한 조회기능을 제공한다.

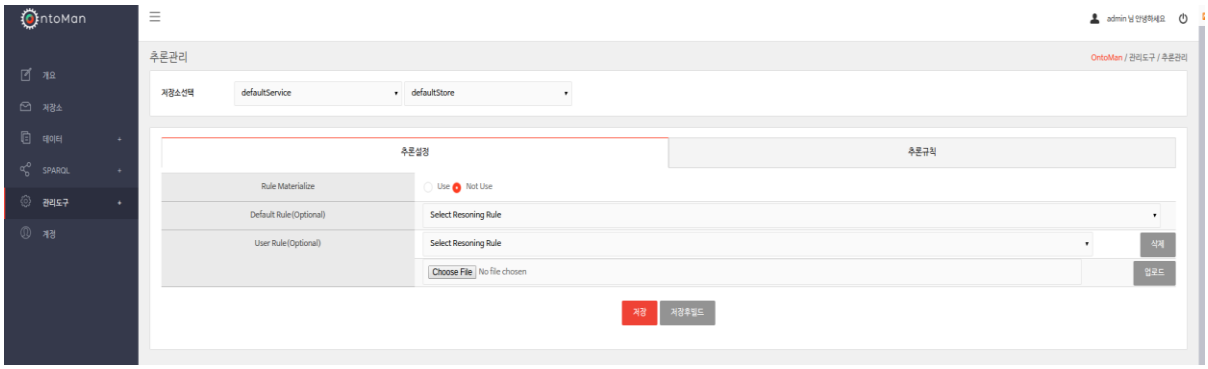
B. 화면 구성 항목

① Select Service - 추론설정을 위한 대상 서비스를 선택한다.

② Select Store – 추론 설정을 수행하기 위한 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택한 대상 서비스에 종속된다.

③ 추론설정 탭 - 스토어 별로 추론 레벨을 설정하는 기능을 수행한다.

④ 추론규칙 탭 - 스토어 별 추론규칙파일을 조회하는 기능을 수행한다.



[4.30 추론설정 메인 화면]

2) 추론설정

A. 추론설정은 스토어별로 설정이 가능하다.

B. 시스템에서 디폴트로 제공되는 추론규칙으로는 rdfs.rules, owl_middle.rules, owl_low.rules. owl_lubm.rules, owl_high.rule 등이 있다.

C. 추론 설정에서 사용자가 직접 작성한 추론규칙을 업로드하여 사용할 수 있으며, 샘플로 sample.rules 규칙파일을 제공한다.

D. 화면 구성 항목

① Select Service - 추론설정을 위한 대상 서비스를 선택한다.

② Select Store - 추론 설정을 수행하기 위한 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택한 대상 서비스에 종속된다.

③ Rule Materialize - 규칙 Materialize 사용여부를 결정 한다.

④ Default Rule(Optional) - 디폴트 추론 규칙파일을 선택한다.

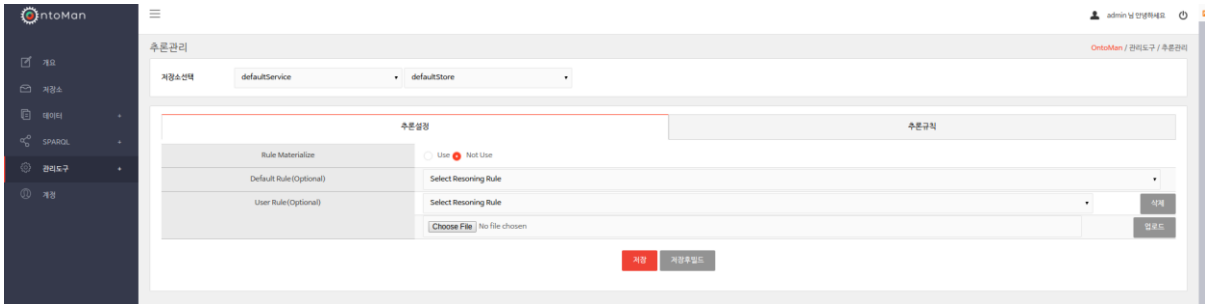
⑤ User Rule(Optional) - 스토어에 업로드된 사용자 정의 규칙 파일이 있으면 사용자 정의 규칙 파일을 선택한다.

⑥ [업로드] 버튼 - 사용자정의 규칙파일을 업로드한다.

⑦ [삭제] 버튼 - 사용자가 업로드한 규칙파일을 삭제한다.

⑧ [저장] 버튼 - 사용자가 설정한 추론 설정을 저장하되 빌드작업을 수행하지 않는다.

⑨ [저장후빌드] 버튼 - 사용자가 설정한 추론 설정을 저장한 뒤 , 빌드작업을 수행한다.



[4.31 추론설정 화면]

3) 추론규칙

A. 추론규칙은 스토어별로 설정이 가능하다.

B. 시스템에서 제공되는 기본규칙과 업로드된 사용자 규칙 파일에 대한 조회기능을 수행한다.

C. 화면 구성 항목

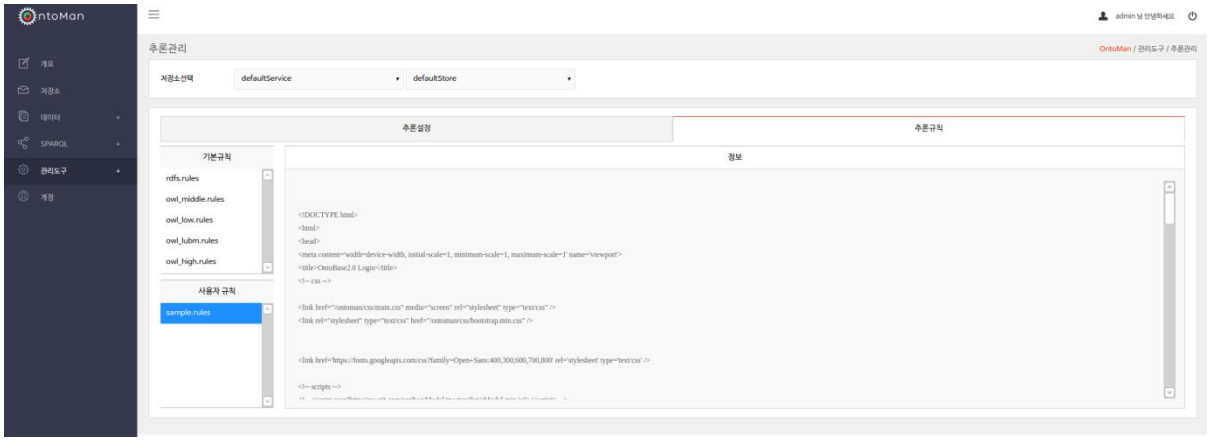
① Select Service - 추론설정을 위한 대상 서비스를 선택한다.

② Select Store - 추론 설정을 수행하기 위한 스토어를 선택한다. 선택할 수 있는 스토어는 앞에서 선택한 대상 서비스에 종속된다.

③ 기본규칙 - 시스템에서 제공하는 규칙파일들을 목록으로 제공한다.

④ 사용자 규칙 - 사용자가 작성한 규칙파일들을 목록으로 제공한다.

⑤ 정보 - 기본규칙 목록과 사용자규칙 목록에서 특정 목록을 클릭하면 해당 파일의 내용을 정보영역에서 보여준다.



[4.32 추론규칙 화면]

3.7 계정

1) 웹 관리도구에 접근할 수 있는 계정을 관리한다.

2) Main

A. 웹 관리도구에 접근할 수 있는 계정을 관리한다.

B. 계정 추가 - [계정등록] 탭을 클릭하면 추가 화면으로 이동한다.

C. 계정 수정 – 사용자목록에서 [수정] 버튼을 클릭하면 수정 화면으로 이동한다.

D. 계정 삭제 – 사용자목록에서 [삭제] 버튼을 클릭하면 삭제할 수 있다.

E. 화면 구성 항목

① 사용자 목록 탭 - 사용자 목록을 조회한다.

② 계정등록 탭 - 계정 등록 기능을 수행한다.

③ No. - 순번

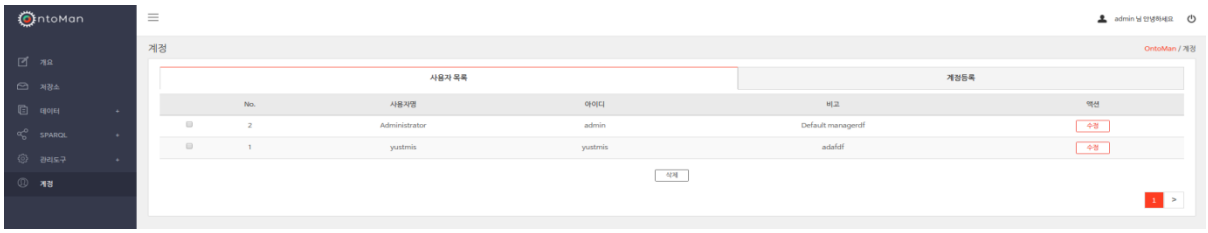
④ 사용자명 - 사용자 이름

⑤ 아이디 - 사용자 아이디

⑥ 비고 - 계정 설명

⑦ [액션] 버튼 - 계정 수정을 수행하는 버튼

⑧ [삭제] 버튼 - 계정 삭제를 수행하는 버튼



[4.33 계정 메인]

3) 계정등록

A. 웹 관리도구에 접근할 수 있는 계정을 추가한다.

B. 화면 구성 항목

① 사용자 아이디 - 새로운 계정 ID

② 사용자 이름 - 사용자 이름

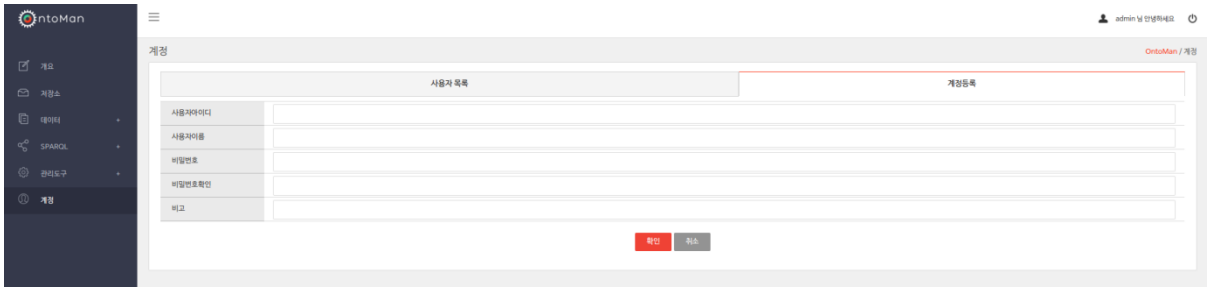
③ 비밀번호 - 계정 암호 설정

④ 비밀번호 확인 - 입력한 암호를 재확인

⑤ 비고 - 계정에 대한 설명

⑥ [확인] 버튼 - 계정 추가 확인 버튼

⑦ [취소] 버튼 - 계정 추가 취소 버튼



[4.34 계정 등록 화면]

4) 계정수정

A. 웹 관리도구에 접근할 수 있는 계정을 수정한다.

B. 화면 구성 항목

① 사용자 아이디 - 사용자 아이디

② 사용자 이름 - 사용자 이름

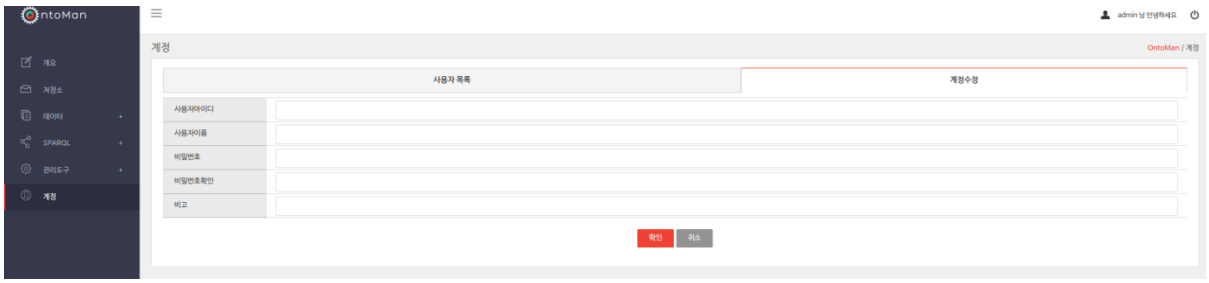
③ 비밀번호 - 암호

④ 비밀번호확인 - 입력한 비밀번호 확인

⑤ 비고 - 설명

⑥ [확인] 버튼 - 계정의 수정을 실행한다.

⑦ [취소] 버튼 - 계정의 수정을 취소하고 이전화면으로 돌아간다.



[4.35 계정 수정 화면]

Chapter 5

OntoBase2.0 Client

Chapter 5. OntoBase2.0 Client

1. OntoBase2.0 Client 개요

1.1 시스템 개요

OntoBase2.0 Client 는 OntoBase2.0 의 클라이언트 모듈로서 클라이언트 API 라이브러리, API Document, 사용 예제 등으로 구성되며 Client API 에는 서버에 데이터를 추가/삭제 및 질의를 생성하고 요청하며, 결과를 받아 처리할 수 있는 로직 등이 포함된다.

2. OntoBase2.0 Client 구성

2.1 클라이언트 API 라이브러리

1) 스토어의 관리, 데이터의 추가, 삭제 및 질의 들을 실행할 수 있는 라이브러리로 java 언어로 구성되어 있다.
2) 사용시에는 아래의 라이브러리들을 import 해서 사용해야만 한다. 아래의 라이브러리들을 통해서, 서버에 데이터의 추가/삭제 및 질의 명령을 수행할 수 있으며, 특히, 이러한 데이터의 추가/삭제를 위해서 사용자가 쉽게 접근 가능하도록 Jena 의 Model 이나 Triple

등을 사용하여 명령을 수행할 수 있는 기능도 제공한다. (기본적으로 라이브러리 목록에는 Jena 라이브러리가 포함되어 있다.)

2.2 API Document

1) 클라이언트 API 라이브러리 Java Document 로서 Jena API 도 포함되어 있다.
2) API Document 목록
   가) OntoBase2.0 Client API Document
   나) Jena API Document

2.3 사용 예제

1) 클라이언트 API 라이브러리를 사용한 예제들이 포함되어 있다.

3. 주요 라이브러리

3.1 StoreManager.java

1) 패키지: com.list.ontobase.client.mgr.StoreManger
2) 스토어의 관리를 담당하는 클래스
3) 트리플 데이터들을 서버에 추가 및 삭제하는 다양한 메서드를 제공한다. (클라이언트에 위치하는 데이터를 서버로 전송하여 추가 및 삭제한다.)
4) 클래스의 주요 메서드는 다음과 같다.

Constructor Summary

Constructors

Constructor and Description
`StoreManager(String id, String pwd, String ip, int port, String service, String store)` constructor

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
int	`addFile(File file)` add triple from file in client
int	`addFiles(String dirName)` add triple from files in client
int	`addModel(com.hp.hpl.jena.rdf.model.Model model)` add triple from JENA model
int	`addTriple(com.hp.hpl.jena.graph.Triple triple)` add triple
int	`addTriple(com.hp.hpl.jena.graph.Triple[] triple)` add triple from triple array
int	`bulkLoading(String dirName)` bulk loading with files in client
int	`deleteFile(File file)` delete triple from file in client
int	`deleteFiles(String dirName)` delete triple from files in client
int	`deleteModel(com.hp.hpl.jena.rdf.model.Model model)` delete triple from JENA model
int	`deleteTriple(com.hp.hpl.jena.graph.Node s, com.hp.hpl.jena.graph.Node p, com.hp.hpl.jena.graph.Node o)` delete triple
int	`deleteTriple(com.hp.hpl.jena.graph.Triple triple)` delete triple
int	`deleteTriple(com.hp.hpl.jena.graph.Triple[] triple)` delete triple from triple array
void	`export(RDFExport exporter, int dataType)` export triple
void	`export(RDFExport exporter, int dataType, int fetchSize)` export triple currently, only support NTRIPLE
int	`getAxiomCount()` get total count of axiom triple
int	`getFactCount()` get total count of fact triple
int	`getInferenceCount()` get total count of inferred triple
com.hp.hpl.jena.rdf.model.Model	`getModel()` get model
PrefixManager	`getPrefixManager()` get PrefixManager
ServerBuildManager	`getServerBuildManager()` get ServerBuildManager
int	`getTotalCount()` get total count of triple
boolean	`initStore()` initialize store
int	`reasonerTextSearchBuild(String workFlag)` reasoner and text search build
com.list.ontobase.msg.inference.rete.verify.VerifyingInfoM[]	`verify(com.hp.hpl.jena.graph.Triple t, boolean recursive)` verify inference if recursive parameter is true, return recursive verify result.
String	`verifyToString(com.hp.hpl.jena.graph.Triple t, boolean recursive)` verify inference if recursive parameter is true, return recursive verify result.

Methods inherited from class com.list.ontobase.client.mgr.AbstractStoreManager

isFinishWork

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

[그림 5.1 StoreManager 의 method]

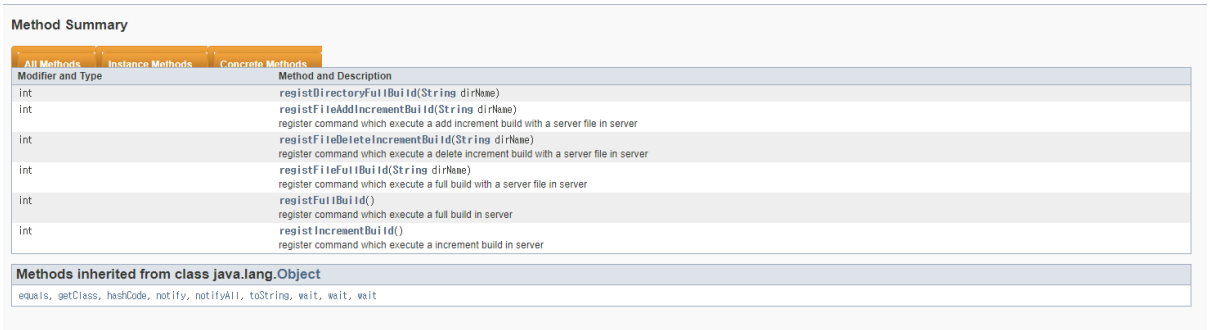
3.2 ServerBuildManager.java

1) 패키지: com.list.ontobase.client.mgr.ServerBuildManger
2) 스토어의 서버 빌드 명령을 담당하는 클래스

3) 서버에서 동작하는 다양한 빌드 명령의 실행을 등록하는 메서드를 제공한다. (StoreManager 에서 제공되는 추가 및 삭제 메서드는 데이터가 클라이언트로부터 서버로 전송되어 실행되는 형태이고, 여기서는 서버 자체적으로 OntoTrans2.0 으로 연계되어 빌드하거나 또는 수동으로 서버에 존재하는 파일을 빌드할 경우에 사용된다. 특히, 벌크 빌드 같은 경우에는 서버에 파일을 로딩해서 서버에서 실행되도록 구성되어 있다.)

4) 기본 제공 아이디는 admin, 패스워드는 admin@#$ 이다.

5) 클래스의 주요 메서드는 다음과 같다.



[그림 5.1 ServerBuildManager 의 method]

3.3 SparqlQuery.java

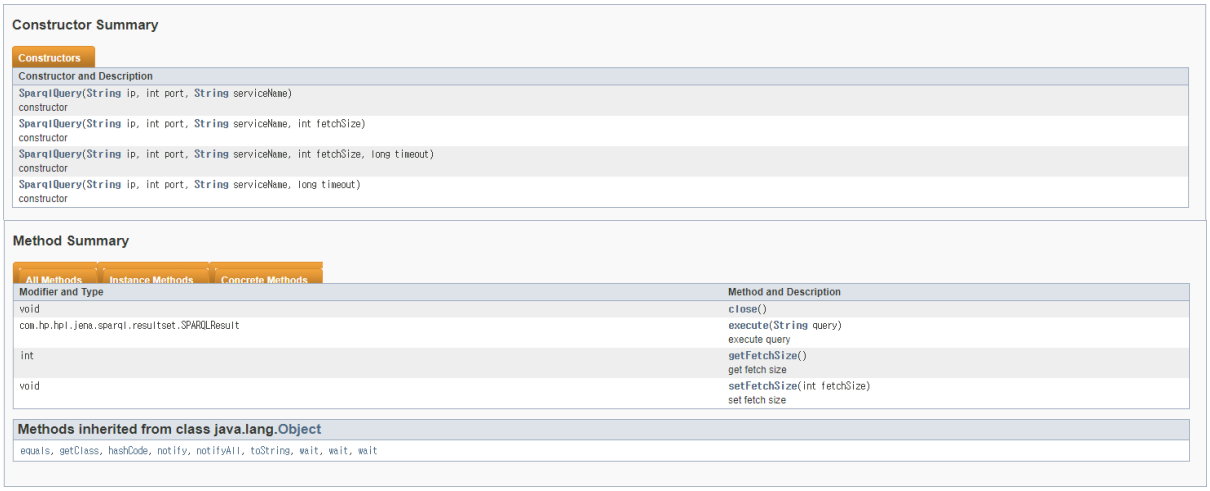
1) 패키지: com.list.ontobase.client.query.SparqlQuery

2) 쿼리를 담당하는 클래스

3) 서버에 SPARQL 질의를 실행하고 결과를 받는다. 기본적으로 SPARQL 질의의 4 가지 형태인 SELECT, ASK, DESCRIBE, CONSTRUCT 형태를 모두 지원한다.

4) OntoBase2.0 의 쿼리엔진은 기본적으로 ARQ 쿼리 엔진을 기본으로 구현되었기 때문에 보다 자세한 스펙은 ARQ 2.8.7 버전을 참조하기 바란다.

5) 클래스의 주요 메서드는 다음과 같다. (fetchSize 는 서버로부터 결과를 가져올 때, 전송 개수를 말하며, 생성자에서 개수를 설정할 수 있다. 디폴트 fetchSize 는 10 만개이다.)



Constructor Summary

Constructors

Constructor and Description
SparqlQuery(String ip, int port, String serviceName) constructor
SparqlQuery(String ip, int port, String serviceName, int fetchSize) constructor
SparqlQuery(String ip, int port, String serviceName, int fetchSize, long timeout) constructor
SparqlQuery(String ip, int port, String serviceName, long timeout) constructor

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	close()
com.hp.hpl.jena.sparql.resultset.SPARQLResult	execute(String query) execute query
int	getFetchSize() get fetch size
void	setFetchSize(int fetchSize) set fetch size

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

[그림 5.2 SparqlQuery 의 method]

6) 쿼리

가) Select 쿼리

```java
/**
 * Sample Select Query
 *
 * @author Administrator
 */
public class SelectQuery {
    public static void main(String[] args) throws Exception {
        // create SparqlQuery
        String ip = "localhost";
        int port = 9999;
        String serviceName = "defaultService";

        SparqlQuery sq = null;
        try {
            sq = new SparqlQuery(ip, port, serviceName);

            // create query string
            StringBuffer sb = new StringBuffer();
            sb.append(" PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>");
            sb.append(" PREFIX owl: <http://www.w3.org/2002/07/owl#>");
            sb.append(" SELECT *");
            sb.append(" WHERE");
            sb.append(" {");
            sb.append("  ?s rdf:type owl:Class .");
            sb.append(" }");

            // execute query
            SPARQLResult result = sq.execute(sb.toString());
            if (result == null) {
                throw new Exception("Error: SPARQLResult is null");
            } else {
                ResultSet rs = result.getResultSet();
                int count = 0;
                for (; rs.hasNext();) {
                    QuerySolution rb = rs.nextSolution();
                    System.out.println(rb.toString());
                    count++;
                }
                System.out.println(">>> Result count=" + count);
            }
        } finally {
            if (sq != null) {
                sq.close();
            }
        }
    }
}
```

[그림 5.3 Select Query 의 예제]

나) Ask 쿼리

```java
/**
 * Sample Ask Query
 *
 * @author Administrator
 */
public class AskQuery {
    public static void main(String[] args) throws Exception {
        // create SparqlQuery
        String ip = "localhost";
        int port = 9999;
        String serviceName = "defaultService";

        SparqlQuery sq = null;
        try {
            sq = new SparqlQuery(ip, port, serviceName);

            // create query string
            StringBuffer sb = new StringBuffer();
            sb.append(" PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>");
            sb.append(" PREFIX owl: <http://www.w3.org/2002/07/owl#>");
            sb.append(" ASK");
            sb.append(" WHERE");
            sb.append(" {");
            sb.append("  ?s rdf:type owl:Class .");
            sb.append(" }");

            // execute query
            SPARQLResult result = sq.execute(sb.toString());
            boolean re = result.getBooleanResult();
            System.out.println(">>> result=" + re);
        } finally {
            if (sq != null) {
                sq.close();
            }
        }
    }
}
```

[그림 5.4 Ask Query 의 예제]

다) Construct 쿼리

```java
/**
 * Sample Construct Query
 *
 * @author Administrator
 */
public class ConstructQuery {
    public static void main(String[] args) throws Exception {
        // create SparqlQuery
        String ip = "localhost";
        int port = 9999;
        String serviceName = "defaultService";

        SparqlQuery sq = null;
        try {
            sq = new SparqlQuery(ip, port, serviceName);

            // create query string
            StringBuffer sb = new StringBuffer();
            sb.append(" PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>");
            sb.append(" PREFIX owl: <http://www.w3.org/2002/07/owl#>");
            sb.append(" PREFIX wine:<http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>");
            sb.append(" CONSTRUCT ");
            sb.append(" {");
            sb.append(" <http://test/testWine> wine:locatedIn ?o");
            sb.append(" }");
            sb.append(" {");
            sb.append("  ?s rdf:type owl:Class .");
            sb.append(" }");

            // execute query
            SPARQLResult result = sq.execute(sb.toString());
            Model model = result.getModel();
            if (model == null)
                System.out.println("null");
            else {
                model.write(new PrintWriter(System.out));
            }
        } finally {
            if (sq != null) {
                sq.close();
            }
        }
    }
}
```

[그림 5.5 Construct Query 의 예제]

라) Describe 쿼리

```java
/**
 * Sample Describe Query
 *
 * @author Administrator
 */
public class DescribeQuery {
    public static void main(String[] args) throws Exception {
        // create SparqlQuery
        String ip = "localhost";
        int port = 9999;
        String serviceName = "defaultService";

        SparqlQuery sq = null;
        try {
            sq = new SparqlQuery(ip, port, serviceName);

            // create query string
            StringBuffer sb = new StringBuffer();
            sb.append(" PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>");
            sb.append(" PREFIX owl: <http://www.w3.org/2002/07/owl#>");
            sb.append(" DESCRIBE ?s");
            sb.append(" {");
            sb.append("  ?s rdf:type owl:Class .");
            sb.append(" }");

            // execute query
            SPARQLResult result = sq.execute(sb.toString());
            Model model = result.getModel();
            if (model == null)
                System.out.println("null");
            else {
                model.write(System.out);
            }
        } finally {
            if (sq != null) {
                sq.close();
            }
        }
    }
}
```

[그림 5.6 Describe Query 의 예제]

4. 지원 옵션 및 사용 예제

다음은 SPARQL 질의를 위한 다양한 옵션 및 사용 예제 들이다. 좀더 자세한 사항은 SPARQL 스펙 및 ARQ 문서를 찾아 보길 권장한다.

4.1 SPARQL : Basic Graph Patterns

SPARQL 에서는 기본적인 Graph Pattern 을 사용하여 질의를 수행한다. Graph Pattern 은 트리플 형태의 패턴으로서 변수 또는 노드가 올 수 있다. 변수의 경우는 "?변수명" 과 같은 형태로 표현한다. 사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> . _:a foaf:name "Alice" . _:b foaf:name "Bob" .

Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/>
	
	SELECT ?x ?name
	
	WHERE
	
	{
	
	?x foaf:name ?name .
	
	}

[Query Result]

x	name
_:c	"Alice"
_:d	"Bob"

4.2 SPARQL : Group Graph Patterns

SPARQL 에서는 기본적인 Graph Pattern 들의 그룹을 사용하여 질의를 수행할 수 있다. Group Graph Pattern 은 그룹 내의 Graph Pattern 들이 모두 만족하는 결과를 리턴하며, { } 를 사용해서 그룹을 명시적으로 표현한다. 사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> . _:a foaf:name "Johnny Lee Outlaw" . _:a foaf:mbox <mailto:jlow@example.com> . _:b foaf:name "Peter Goodguy" . _:b foaf:mbox <mailto:peter@example.org> .
Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/> SELECT ?name ?mbox WHERE { ?x foaf:name ?name . ?x foaf:mbox ?mbox }

[Query Result]

name	mbox
"Johnny Lee Outlaw"	<mailto:jlow@example.com>
"Peter Goodguy"	<mailto:peter@example.org>

4.3 SPARQL : Variable Constraints

SPARQL 에서는 Graph Pattern 내의 변수에 대해서 FILTER 옵션을 사용해서 제약을 줄 수 있다. 기본적으로 SPARQL 에서는 다양한 operator 들을 제공한다. 주요 operator 는 다음과 같다. (더 자세한 내용은 SPARQL 스펙을 참조)

[주요 Unary Operators]

Operator	Type(A)	Result Type
!A	xsd:boolean	xsd:boolean
+A	numeric	numeric
-A	numeric	numeric
BOUND(A)	Variable	xsd:boolean
isIRI(A)	RDF Term	xsd:boolean
isURI(A)	RDF Term	xsd:boolean
isBlank(A)	RDF Term	xsd:boolean
isLiteral(A)	RDF Term	xsd:boolean

STR(A)	literal	simple literal
STR(A)	IRI	simple literal
LANG(A)	literal	simple literal
DATATYPE(A)	typed literal	IRI
DATATYPE(A)	simple literal	IRI

[주요 Binary Operators]

Operator	Type(A)	Type(B)	Result Type
A || B	xsd:boolean	xsd:boolean	xsd:boolean
A && B	xsd:boolean	xsd:boolean	xsd:boolean
A = B	numeric	numeric	xsd:boolean
A = B	simple literal	simple literal	xsd:boolean
A = B	xsd:string	xsd:string	xsd:boolean
A = B	xsd:boolean	xsd:boolean	xsd:boolean
A = B	xsd:datetime	xsd:datetime	xsd:boolean
A != B	numeric	numeric	xsd:boolean
A != B	simple literal	simple literal	xsd:boolean
A != B	xsd:string	xsd:string	xsd:boolean
A != B	xsd:boolean	xsd:boolean	xsd:boolean
A != B	xsd:datetime	xsd:datetime	xsd:boolean

A < B	numeric	numeric	xsd:boolean
A < B	simple literal	simple literal	xsd:boolean
A < B	xsd:string	xsd:string	xsd:boolean
A < B	xsd:boolean	xsd:boolean	xsd:boolean
A < B	xsd:datetime	xsd:datetime	xsd:boolean
A > B	numeric	numeric	xsd:boolean
A > B	simple literal	simple literal	xsd:boolean
A > B	xsd:string	xsd:string	xsd:boolean
A > B	xsd:boolean	xsd:boolean	xsd:boolean
A > B	xsd:datetime	xsd:datetime	xsd:boolean
A <= B	numeric	numeric	xsd:boolean
A <= B	simple literal	simple literal	xsd:boolean
A <= B	xsd:string	xsd:string	xsd:boolean
A <= B	xsd:boolean	xsd:boolean	xsd:boolean
A <= B	xsd:datetime	xsd:datetime	xsd:boolean
A >= B	numeric	numeric	xsd:boolean
A >= B	simple literal	simple literal	xsd:boolean
A >= B	xsd:string	xsd:string	xsd:boolean
A >= B	xsd:boolean	xsd:boolean	xsd:boolean
A >= B	xsd:datetime	xsd:datetime	xsd:boolean

A * B	numeric	numeric	numeric
A / B	numeric	numeric	numeric
A + B	numeric	numeric	numeric
A - B	numeric	numeric	numeric
A = B	RDF Term	RDF Term	xsd:boolean
A != B	RDF Term	RDF Term	xsd:boolean
sameTERM(A)	RDF Term	RDF Term	xsd:boolean
langMATCHES(A, B)	simple literal	simple literal	xsd:boolean
REGEX(String, Pattern)	simple literal	simple literal	xsd:boolean

[주요 Trinary Operators]

Operator	Type(A)	Type(B)	Type(C)	Result Type
REGEX(String, Pattern, FLAGS)	simple literal	simple literal	simple literal	xsd:boolean

사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix dc: <http://purl.org/dc/elements/1.1/> . @prefix : <http://example.org/book/> . @prefix ns: <http://example.org/ns#> .

	:book1 dc:title "SPARQL Tutorial" . :book1 ns:price 42 . :book2 dc:title "The Semantic Web" . :book2 ns:price 23 .
Query	PREFIX dc: <http://purl.org/dc/elements/1.1/> PREFIX ns: <http://example.org/ns#> SELECT ?title ?price WHERE { ?x ns:price ?price . FILTER (?price < 30) ?x dc:title ?title . }

[Query Result]

title	price
"The Semantic Web"	23

4.4 SPARQL : Optional Pattern Matching

SPARQL 에서는 다양한 쿼리결과 생성을 위해서 Optional 키워드를 제공한다. Optional 은 Optional 부분이 매칭된다면 bind 된 결과를 생성하고, 매칭되지 않는다면 bind 가 되지 않은 빈 결과를 생성한다. 사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix dc: <http://purl.org/dc/elements/1.1/> . @prefix : <http://example.org/book/> . @prefix ns: <http://example.org/ns#> . :book1 dc:title "SPARQL Tutorial" . :book1 ns:price 42 . :book2 dc:title "The Semantic Web" . :book2 ns:price 23 .
Query	PREFIX dc: <http://purl.org/dc/elements/1.1/> PREFIX ns: <http://example.org/ns#> SELECT ?title ?price WHERE { ?x dc:title ?title . OPTIONAL { ?x ns:price ?price . FILTER (?price < 30) } }

[Query Result]

title	price
"SPARQL Tutorial"	
"The Semantic Web"	23

4.5 SPARQL : Matching Alternatives

SPARQL 에서는 다양한 Graph Pattern 들의 결과를 조합할 수 있는 기능을 제공하는 UNION 키워드를 제공한다. 사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix dc10: <http://purl.org/dc/elements/1.0/> . @prefix dc11: <http://purl.org/dc/elements/1.1/> . _:a dc10:title "SPARQL Query Language Tutorial" . _:b dc11:title "SPARQL Protocol Tutorial" . _:c dc10:title "SPARQL" . _:c dc11:title "SPARQL (updated)" .
Query	PREFIX dc10: <http://purl.org/dc/elements/1.0/> PREFIX dc11: <http://purl.org/dc/elements/1.1/> SELECT ?x ?y WHERE { { ?book dc10:title ?x } UNION { ?book dc11:title ?y } }

[Query Result]

x	y
	"SPARQL (updated)"

	"SPARQL Protocol Tutorial"
"SPARQL"	
"SPARQL Query Language Tutorial"	

4.6 SPARQL : Sequences and Modifiers – Order By

SPARQL 에서는 Order by 키워드를 통해서 결과의 순서를 결정할 수 있다. 또한 desc(), asc() 의 optional order modifier를 통해서 정렬 형태를 결정할 수 있다. 또한 특정 변수에 bind 된 결과에 다양한 형태가 올 수 있는데, SPARQL 에서 RDF Term 사이의 정렬 순서는 아래와 같은 순서로 고정되어 있다.

A. 할당된 값이 없는 경우

B. Blank Node

C. IRIs

D. RDF Literal

사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix dc: <http://purl.org/dc/elements/1.1/> . @prefix : <http://example.org/book/> . @prefix ns: <http://example.org/ns#> . :book1 dc:title "SPARQL Tutorial" .

	:book1 ns:price 42 . :book2 dc:title "The Semantic Web" .
Query	PREFIX dc: <http://purl.org/dc/elements/1.1/> SELECT ?name WHERE { ?x dc:title ?name } ORDER BY ?name

[Query Result]

name
"SPARQL Tutorial"
"The Semantic Web"

4.7 SPARQL : Sequences and Modifiers – Distinct

SPARQL 에서는 Distinct 키워드를 통해서 중복 결과를 제거할 수 있다. 사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> . _:x foaf:name "Alice" . _:x foaf:mbox <mailto:alice@example.com> .

	_:y foaf:name "Alice" . _:y foaf:mbox <mailto:asmith@example.com> .
Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/> SELECT distinct ?name WHERE { ?x foaf:name ?name }

[Query Result]

name
"Alice"

4.8 SPARQL : Sequences and Modifiers – Offset and Limit

SPARQL 에서는 Offset 과 Limit 키워드를 통해서 결과의 개수를 제한할 수 있다. 사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> . _:x foaf:name "Alice" . _:y foaf:name "Peter" . _:z foaf:name "James" . _:k foaf:name "Robert" .

Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/> SELECT ?name WHERE { ?x foaf:name ?name } ORDER BY ?name OFFSET 1 LIMIT 2

[Query Result]

name
"James"
"Peter"

4.9 SPARQL : RDF Datasets - Graph

SPARQL 에서는 다양한 RDF Dataset 으로 부터 질의가 가능하다. OntoBase2.0 에서는 Service 와 Store 라는 명시적인 구조를 통해서 RDF Dataset 을 지원한다. 동일한 Service 내의 모든 Store 에 대해서 질의가 가능하며, 각각의 RDF Dataset 은 Store 의 구조와 매핑된다. 또한 각 Service 에는 기본 Store 가 존재해서 특별한 RDF Dataset 을 지정하지 않는 경우는 기본 Store 를 대상으로 질의를 수행한다. 각각의 Store 의 Graph Name 은 다음과 같은 구조로 정의되어 있다. : <file:///ontobase/{Service 명}/{Store 명}>

사용 예제는 아래와 같다.

[Example Data & Query]

Data	# Default Graph – defaultService/defaultStore # Graph Name – <file:///ontobase/ defaultService/defaultStore> @prefix foaf: <http://xmlns.com/foaf/0.1/> . _:x foaf:name "Alice" . _:x foaf:name "Peter" . # Other Graph – defaultService/test # Graph Name – <file:///ontobase/ defaultService/test> @prefix foaf: <http://xmlns.com/foaf/0.1/> . _:y foaf:name "Peter" .
Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/> SELECT ?name WHERE { ?x foaf:name ?name Graph <file:///ontobase/ defaultService/test> {?x foaf:name ?name} }

[Query Result]

name
"Peter"

4.10 ARQ Extension Option : All Variable

ARQ 에서는 "*" 키워드를 사용하여 Where 절에 사용된 모든 변수에 대한 결과를 가져올 수 있는 기능을 제공한다. 사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> . _:x foaf:name "Alice" . _:y foaf:name "Peter" . _:z foaf:name "James" . _:k foaf:name "Robert" .
Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/> SELECT * WHERE { ?x foaf:name ?name }

[Query Result]

x	name
_:v1	"Alice"
_:v2	"Peter"
_:v3	"James"
_:v4	"Robert"

4.11 ARQ Extension Option : Count

ARQ 에서는 "count" 키워드를 사용하여 변수에 대한 결과 개수를 가져올 수 있는 기능을 제공한다. 사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> . _:x foaf:name "Alice" . _:y foaf:name "Peter" . _:z foaf:name "James" . _:k foaf:name "Robert" .
Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/> SELECT (count(*) as ?cnt) WHERE { ?x foaf:name ?name }

[Query Result]

cnt
4

4.12 ARQ Extension Option : Group By

ARQ 에서는 결과를 특정 변수에 대해서 Grouping 할 수 있도록 Group By 키워드를 제공한다. 또한 Having 옵션 키워드를 통해서 제약이 가능하다. 사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> . _:a foaf:name "Johnny Lee Outlaw" . _:a foaf:mbox <mailto:jlow@example1.com> . _:a foaf:mbox <mailto:jlow@example2.com> . _:b foaf:name "Peter Goodguy" . _:b foaf:mbox <mailto:peter@example3.org> .
Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/> SELECT ?name WHERE { ?x foaf:name ?name .

	?x foaf:mbox ?mbox } GROUP BY ?name HAVING (count(?mbox) > 1)

[Query Result]

name
"Johnny Lee Outlaw"

4.13 ARQ Extension Option : Let

ARQ 에서는 특정 변수에 대해서 임의로 값을 할당 가능하도록 Let 키워드를 제공한다. 사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> . _:x foaf:name "Alice" . _:y foaf:name "Peter" .
Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/> SELECT ?name, ?z WHERE { ?x foaf:name ?name

	let (?z := 1) }

[Query Result]

name	z
"Alice"	1
"Peter"	1

4.14 ARQ Extension Option : Sub Query

ARQ 에서는 동적으로 다양한 질의 생성이 가능하도록 Sub Query 구문을 지원한다. 사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> . _:a foaf:name "Johnny Lee Outlaw" . _:a foaf:mbox <mailto:jlow@example1.com> . _:a foaf:mbox <mailto:jlow@example2.com> . _:b foaf:name "Peter Goodguy" . _:b foaf:mbox <mailto:peter@example3.org> .
Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/>

	SELECT ?name WHERE { ?x foaf:name ?name . {SELECT ?x WHERE {?x foaf:mbox ?mbox}} }

[Query Result]

name
"Peter Goodguy"
"Johnny Lee Outlaw"
"Johnny Lee Outlaw"

4.15 ARQ Extension Option : Filter Functions

ARQ 는 다양하고 복잡한 쿼리 처리를 위해서 기본적인 SPARQL operator 외에 다양한 function 라이브러리들을 제공한다. 사용자는 ARQ function 라이브러리와 XPath 및 XQuery function 라이브러리를 사용할 수 있다. ARQ function 라이브러리의 네임스페이스는 <http://jena.hpl.hp.com/ARQ/function#> 이고, XPath 및 XQuery function 라이브러리의 네임스페이스는 <http://www.w3.org/2005/xpath-function#> 이다. 또한 일반적으로 ARQ function 라이브러리의 프리픽스는 afn 을 사용하고 XQuery function 라이브러리의 네임스페이스는 fn 을 사용한다.

사용 가능한 Function 라이브러리 목록은 다음과 같다.

[String Functions]

Function Name	Description
afn:substr(*string, startIndex [,endIndex*])	string 의 substring 을 리턴한다. (주의-인덱스는 0부터 시작함)
afn:substring(*string, startIndex [,endIndex*])	afn:substr 과 동일하다.
afn:strjoin(*sep*, *string* ...)	string 들을 sep 구분자를 가지고 연결한 결과를 리턴한다.
afn:sha1sum(*resource*)	Literal 이나 URI 의 SHA1 checksum 을 계산한 결과를 리턴한다.
fn:contains(*string*, *substr*)	string 이 substr 을 포함하는지 여부를 테스트한다.
fn:starts-with(*string*, *match*)	string 이 match 로 시작되는지 여부를 테스트한다.
fn:ends-with(*string*, *match*)	string 이 match 로 끝나는지 여부를 테스트한다.
fn:string-length(*string)*	string 의 길이를 리턴한다.
fn:lower-case(*string)*	string 을 lower-case 해서 리턴한다.
fn:upper-case(*string)*	string 을 upper-case 해서 리턴한다.
fn:matches(*string*, *pattern* [, *flags])*	string 에 대해서 Regular Expression Match 를 테스트한다.
fn:concat(*string*, ...)	string 들을 연결한 결과를 리턴한다.
fn:substring(*string*, *begin* [,*length*])	string 의 substring 을 리턴한다.

	(주의-인덱스는 1부터 시작함)

[Mathematical Functions]

Function Name	Description
afn:min(num1, num2)	두 개의 Number 중에 작은 것을 찾아 리턴한다.
afn:max(num1, num2)	두 개의 Number 중에 큰 것을 찾아 리턴한다.
afn:pi()	PI 값을 XSD Double 형으로 리턴한다.
afn:e()	E 값을 XSD Double 형으로 리턴한다.
fn:round(v)	Number v 의 round 값을 리턴한다. (가장 가까운 integer 값)
fn:abs(v)	Number v 의 abs 값을 리턴한다. (절대값)
fn:floor(v)	Number v 의 floor 값을 리턴한다. (v보다 작은 최대 integer 값)
fn:ceiling(v)	Number v 의 ceiling 값을 리턴한다. (v보다 큰 최소 integer 값)

[Boolean Functions]

Function Name	Description

fn:boolean(*value*)	value 의 유효한 Boolean 값을 리턴한다.
fn:not(*value*)	value 의 유효한 Boolean 값의 negation 값을 리턴한다.

[RDF Graph Functions]

Function Name	Description
afn:bnode(?x)	변수 x 가 blank node 이면 blank node label 을 리턴한다.
afn:localname(?x)	변수 x 가 IRI 라면 localname 을 리턴한다.
afn:namespace(?x)	변수 x 가 IRI 라면 namespace 를 리턴한다.

[Miscellaneous Functions]

Function Name	Description
afn:now()	현재 시간을 XSD datatime 형으로 리턴한다.

사용 예제는 아래와 같다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> . @prefix afn: <http://jena.hpl.hp.com/ARQ/function#> . _:x foaf:name "Alice" .

Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/>
	
	SELECT ?name, ?z
	
	WHERE {
	
	?x foaf:name ?name
	
	let (?z := afn:now())
	
	}

[Query Result]

name	z
"Alice"	"2011-08-05T01:39:24.484Z"^^xsd:dateTime

4.16 ARQ Extension Option : Filter Forms - IF

Filter Form 은 function 과 유사한 operator 로서 Filter 나 LET(assignments), Select 구문에서 사용 가능하다. ARQ 에서는 크게 IF와 COALEASE 의 2가지 종류의 Form 을 제공한다.

IF Form 은 3개의 Argument 를 가지는데, 첫 번째 아규먼트의 Boolean 결과에 따라서 결과가 true 이면 두 번째 아규먼트를 리턴하고, false 이면 세 번째 아규먼트를 리턴한다.

다음은 IF 에 대한 예제이다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> .

	@prefix fn: <http://www.w3.org/2005/xpath-function#> . _:x foaf:name "Alice" . _:y foaf:name "Peter" .

Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/>
	
	SELECT ?name, ?z
	
	WHERE {
	
	?x foaf:name ?name .
	
	let (?z := if(fn:boolean(1), 10, 100))
	
	}

[Query Result]

name	z
"Alice"	10
"Peter"	10

4.17 ARQ Extension Option : Filter Forms - COALESCE

COALEASE Form 은 여러 개의 변수 Argument 를 가지는데, 변수 아규먼트 리스트 중에서 가장 먼저 특정 값으로 bound 되는 변수의 값을 리턴한다.

다음은 COALEASE 에 대한 예제이다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> . _:x foaf:name "Alice" . _:y foaf:name "Peter" .

Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/>
	
	SELECT ?name, ?z
	
	WHERE {
	
	?x foaf:name ?name .
	
	let (?z := coalesce(?name, ?x))
	
	}

[Query Result]

name	z
"Alice"	"Alice"
"Peter"	"Peter"

4.18 ARQ Extension Option : Property Paths

ARQ 에서는 다양한 질의문 생성이 가능하도록 유용한 기능인 Property Path 를 제공한다. Property Path 는 두 그래프 노드 사이의 모든 가능한 경로를 말한다. 예를 들어 일반적인 트리플 패턴의 경우는 길이가 1인 Property Path 이다. ARQ 에서는 이 Property Path 를 사용하여 질의구분에 적용할 수 있다. 예를 들어 foaf:knows/foaf:name 과 같은 Property Path 는 하나의

"knows" 단계와 연결된 사람의 name 정보를 의미한다.

Property Path 에 대한 다양한 사용법은 다음과 같다.

[Property Path Syntax]

Syntax Form	설명
uri	URI 또는 prefixed name 로서 패스의 길이는 1이다.
^elt	Reverse 패스(object 와 subject 를 서로 바꾸어 적용한다.)
(elt)	Property Path 의 그룹을 표현한다. (Bracket 으로 감싸서 표현한다.)
elt1 / elt2	sequence 패스를 표현(elt1 후에 elt2 가 나오는 패스를 의미한다.)
elt1 ^ elt2	*elt1 /^ elt2* 의 축약 표현(elt1 후에 reverse elt2 가 나오는 패스를 의미한다.)
elt1 | elt2	alternative 패스를 표현(elt1 또는 elt2 패스 –모든 가능한 패스)
*elt**	1개 이상 존재하는 elt 패스
elt+	0개 이상 존재하는 elt 패스
elt?	0 또는 1개 존재하는 elt 패스
elt{n,m}	n 과 m 사이의 개수가 존재하는 elt 패스
elt{n}	n개 존재하는 elt 패스
elt{n,}	n개 이상 존재하는 elt 패스

elt{,n}	n개 이하 존재하는 elt 패스
!uri	uri 가 아닌 모든 패스
!(uri1|...|uriN)	uri 리스트 들이 아닌 모든 패스

다음은 Property Path 에 대한 예제이다.

[Example Data & Query]

Data	@prefix foaf: <http://xmlns.com/foaf/0.1/> . _:a foaf:name "Johnny Lee Outlaw" . _:a foaf:mbox <mailto:jlow@example1.com> . _:a foaf:mbox <mailto:jlow@example2.com> . _:b foaf:name "Peter Goodguy" . _:b foaf:mbox <mailto:peter@example3.org> .

Query	PREFIX foaf: <http://xmlns.com/foaf/0.1/>
	
	SELECT ?x, ?z
	
	WHERE {
	
	?x !foaf:mbox ?z .
	
	}

[Query Result]

x	z
_:v1	"Johnny Lee Outlaw"
_:v1	"Peter Goodguy"

Chapter 6

TroubleShooting

Chapter 6. TroubleShooting

1. OntoBase2.0 Server

1.1 Q-1: 서버 구동 시 메모리가 부족합니다. (OutOfMemory 에러)

1) 원인

가) 서버구동 시 필요한 메모리가 부족한 경우에 발생한다.

2) 해결책

가) 서버구동 시 충분한 메모리를 할당하여 준다.

나) $SERVER_HOME₩bin 디렉토리에 startupServer 파일을 연다.

다) Java 실행 옵션 중에 –Xms512m –Xmx1024m 과 같이 메모리 크기를 확장시켜준다.

A. -Xms 할당하려는 최소크기의 메모리

B. –Xmx 할당하려는 최대크기의 메모리

라) 여기서 Xmx 값을 더 확장할 수 없는 경우에는 $SERVER_HOME₩setting 아래의 각 스토어 디렉토리의 스토어명.properties 를 열어 캐시크기를 줄여준다. (rep.cacheSize 및 index.cacheSize)

1.2 Q-2: 데이터 빌드 중에 서버 강제 종료 후 에러가 발생합니다.

1) 원인

가) 데이터 빌드 중에 서버를 강제 종료했을 경우에, 내부적인 파일 데이터가 깨져서 에러가 발생한다.

2) 해결책

가) 서버를 종료한다.

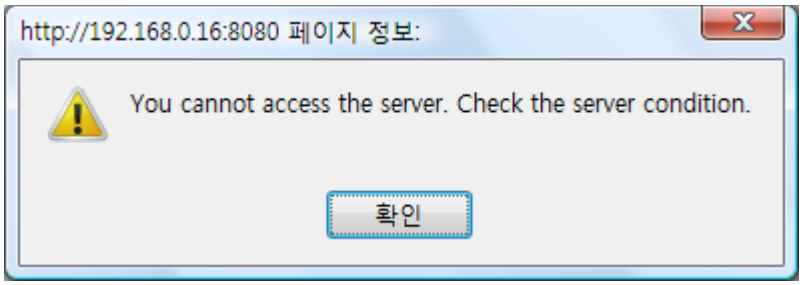
나) $SERVER_HOME₩store 디렉토리의 모든 파일을 강제 삭제한다. (파일이 깨졌기 때문에 스토어 자체를 수동으로 clear 하여야만 한다.)

다) 서버를 재시작한다.

라) 빌드를 재실행한다.

2. OntoBase2.0 Manager

2.1 Q-1: 관리도구 로그인 페이지에 접속 후 다음과 같은 메시지가 나타납니다.



[그림 6.1 로그인 페이지 접속 에러 화면]

1) 원인

가) 관리도구에서 서버로의 접속에 실패한 경우에 발생함

나) 극히 드문 경우이지만 임베디드 된 데이터베이스가 서비스 중이 아닐 경우에도 발생

2) 해결책

가) $MANAGER_HOME/WEB-INF/classes/setting.properties 파일을 확인한다. 관련 속성 중에 database.xx 로 시작되는 항목 중 ip, port 항목을 확인한다. OntoBase2.0 서버는 데이터베이스의 접속 IP 주소와 OntoBase2.0 서버의 IP 주소가 일치하기 때문에 database.connect.ip 주소가 OntoBase2.0 서버가 설치된 아이피 주소와 일치하는지 확인하여야 한다.

나) $MANAGER_HOME/include/setting.jsp 파일을 확인한다. 이곳에서는 OntoBase2.0 관리서버로의 접속 정보를 확인할 수 있다. 만약 OntoBase2.0 관리서버의 manager 포트를 기본 9995 가 아닌 다른 포트번호로 설정하였다면 이 jsp 파일에서 해당 접속포트 번호를 변경해야만 한다.

다) 극히 드문 경우이지만 위의 방법대로도 해결되지 않은 경우 OntoBase2.0 서버의 Database 의 동작 유무를 한다. Ontobase2.0 서버의 DB 가 동작하고 있지 않다면 먼저 $SERVER_HOME₩bin 디렉토리의 shutdownDB 파일을 실행하여 데이터베이스 서비스를 정지한 후 다시 $SERVER_HOME ₩bin 디렉토리의 startupDB 파일을 실행하여 임베디드 된 데이터베이스를 시작한다.

2.2 Q-2: 관리도구의 서버 메뉴의 Synchronization 항목에서 서비스나 스토어 시작 / 종료시 java.lang.NullPointerException 오류가 발생하는 경우

1) 원인

가) OntoBase2.0 서버가 정상적으로 시작되지 않았을 경우 발생

나) 수동항목으로 서비스 혹은 스토어를 추가한 경우 이에 따른 각 서비스와 스토어가 비정상적으로 로드되어 서버 서비스가 올바르게 제공할수 없는 경우 발생

2) 해결책

가) OntoBase2.0 을 재시작하여 서버가 정상적으로 시작이 되는지 확인한다.

나) 서버는 정상적으로 시작되었는데 서비스나 스토어가 계속적으로 관련 오류를 발생하면 관리도구를 통해서 해당 서비스 혹은 스토어를 삭제한다.

다) 관리도구를 통해서 해당 서비스 혹은 스토어를 정상 삭제하고 OntoBase2.0 을 재부팅하여 서버가 정상적으로 시작이 되는지 확인한다.

라) 관리도구를 통해 서비스 혹은 스토어가 정상 삭제되지 않는 경우 사용자는 직접 $ONTOBASE_HOME/setting/store.properties 파일을 편집하여 해당 서비스 및 스토어정보를 삭제한다. 스토어 정보 삭제시 $ONTOBASE_HOME/setting/ 폴더 아래에 생성된 서비스 폴더명과 $ONTOBASE_HOME/store/ 폴더아래에 존재하는 서비스 폴더명을 삭제한다. (이경우 데이터를 다시 빌드하여야 한다.)

마) 상기 방법으로 해결되지 않는 문제에 대해서는 OntoBase2.0 기술 지원팀으로 문의요망.

부 록

부록

A. Rule File

1. 룰 파일의 구조

가) 룰 파일의 구성

✓ Prefix: 룰 파일에서 사용되는 프리픽스 정의

✓ Axiom: 추가할 Axiom 정의

✓ Rule: 적용될 룰 정의(SWRL 문법)

✓ Comment: # 문자로 시작

나) Rule 의 구성

✓ SWRL 을 사용하여 룰을 정의하며 SWRL 의 Human Readable Syntax를 사용하여 각 룰을 정의함. (http://www.w3.org/Submission/SWRL/ 참조)

✓ SWRL Human Readable Syntax

✓ [{Rule Name}: {LHS(Body)} -> {RHS(Head)}]

✓ {Rule Name}: 룰의 이름

✓ {LHS(Body)}: 룰의 antecedent로서, atom 혹은 atom 의 셋으로 구성되며 각 atom 들은 ^ 문자를 사용하여 연결함

✓ {RHS(Head)}: 룰의 consequent로서, atom 혹은 atom 의 셋으로 구성되며 각 atom 들은 ^ 문자를 사용하여 연결함

✓ Example

✓ [rdfs9: rdfs:subClassOf(?x, ?y) ^ ?x(?a) ^ notEqual(?x, ?y) -> ?y(?a)]

✓

```
#------------------------------------------------------------------
# RDFS Closure rules
#------------------------------------------------------------------

#------------------------------------------------------------------
# Prefix
#------------------------------------------------------------------
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

#------------------------------------------------------------------
# Axiom
#------------------------------------------------------------------
axiom rdf:type rdf:type rdf:Property
axiom rdf:type rdfs:domain rdfs:Resource

#------------------------------------------------------------------
# Rule
#------------------------------------------------------------------
[rdfs2: rdfs:domain(?p, ?c) ^ ?p(?x, ?y) -> ?c(?x)]
[rdfs3: rdfs:range(?p, ?c) ^ ?p(?x, ?y) -> ?c(?y)]
[rdfs5: rdfs:subPropertyOf(?a, ?b) ^ rdfs:subPropertyOf(?b, ?c) -> rdfs:subPropertyOf(?a, ?c)]
[rdfs7: rdfs:subPropertyOf(?p, ?q) ^ ?p(?a, ?b) ^ notEqual(?p, ?q) -> ?q(?a, ?b)]
[rdfs8: rdfs:Class(?a) -> rdfs:subClassOf(?a, rdfs:Resource)]
[rdfs9: rdfs:subClassOf(?x, ?y) ^ ?x(?a) ^ notEqual(?x, ?y) -> ?y(?a)]
[rdfs11: rdfs:subClassOf(?a, ?b) ^ rdfs:subClassOf(?b, ?c) -> rdfs:subClassOf(?a, ?c)]
[rdfs12: rdfs:ContainerMembershipProperty(?x) -> rdfs:subPropertyOf(?x, rdfs:member)]
[rdfs13: rdfs:Datatype(?x) -> rdfs:subClassOf(?x, rdfs:Literal)]
```

Prefix 정의

Axiom 정의

Rule 정의: SWRL 룰

Rule Name

LHS(Body)

RHS(Head)

✓

✓ [그림 부록 A.1 룰 파일의 구성]

✓

✓ 시스템 제공 룰 파일 – 시스템에서 제공하는 기본적인 룰 파일은 아래와 같이 4개를 제공하며 서비스 생성 시에 각 추론 수준 별로 선택해서 사용 가능하다. (각 추론 수

준에 대한 내용은 아래의 [부록 A-2 시스템 룰 파일] 부분을 참조.)

- ✓ rdfs.rules
- ✓ owl_low.rules
- ✓ owl_middle.rules
- ✓ owl_high.rules

다) 사용자 정의 룰 파일

- ✓ 사용자는 시스템에서 제공되는 룰 파일 외에 직접 필요한 룰들을 정의해서 사용 가능하다. 실제 룰들을 정의 하는 방법은 위의 [부록 A-1-가] 룰 파일의 구조]를 참조
- ✓ 실제 Rule 들은 SWRL 의 룰 규칙을 따른다.

2. 시스템 룰 파일

가) 시스템에서 제공하는 기본적인 룰 파일은 각 추론 수준 별로 4개의 파일을 제공한다.

- ✓ rdfs.rules
- ✓ owl_low.rules
- ✓ owl_middle.rules
- ✓ owl_high.rules

나) 시스템에서 제공하는 기본적인 룰 파일의 각 추론 수준은 아래와 같다.

Type	Rule Name	rdfs.rules	owl_low.ru	owl_middle	owl_high.r

			les	.rules	ules
rdfs	rdf1_rdfs4a_4b	O	-	-	-
	rdfs2	O	O	O	O
	rdfs3	O	O	O	O
	rdfs5	O	O	O	O
	rdfs7	O	O	O	O
	rdfs8	O	-	-	-
	rdfs9	O	O	O	O
	rdfs11	O	O	O	O
	rdfs12	O	O	O	O
	rdfs13	O	O	O	O
	rdfs_ext1	O	O	O	O
	rdfs_ext2	O	O	O	O
	rdfs_ext3	O	O	O	O
	rdfs_ext4	O	O	O	O
thing	thing1	-	O	O	O
sameAs	sameAs1	-	O	O	O
	sameAs2	-	O	O	O
	sameAs3	-	-	O	O
	sameAs4	-	-	O	O

	sameAs5	-	-	O	O
inverseOf	inverseOf1	-	O	O	O
	inverseOf11	-	O	O	O
	inverseOf2	-	-	O	O
	inverseOf3	-	-	O	O
	inverseOf4	-	-	O	O
symmetricProperty	symmetricProperty1	-	O	O	O
	symmetricProperty2	-	-	O	O
functionalProperty	functionalProperty1	-	O	O	O
	functionalProperty2	-	-	O	O
inverseFunctional	inverseFunctional1	-	O	O	O
	inverseFunctional2	-		O	O
transitive	transitiveProperty1	-	O	O	O
equivalent	equivalentClass1	-	O	O	O
	equivalentClass2	-	O	O	O
	equivalentProperty1	-	O	O	O
	equivalentProperty2	-	O	O	O

allValuesFrom	allValuesFrom1	-	O	O	O
	allValuesFrom2	-	-	-	O
someValuesFrom	someValuesFrom1	-	O	O	O
	someValuesFrom2	-	-	-	O
allDifferent	allDifferent1	-	-	-	O
	allDifferent2	-	-	-	O
cardinality	minCardinality1	-	-	-	O
	minCardinality2	-	-	-	O
	maxCardinality1	-	-	-	O
oneOf	oneOf1	-	-	-	O
	oneOf2	-	-	-	O
unionOf	unionOf1	-	-	-	O
	unionOf2	-	-	-	O
hasValue	hasValue1	-	O	O	O
	hasValue2	-	O	O	O
	hasValue3	-	-	-	O
	hasValue4	-	-	-	O
disjointWith	disjointWith1	-	-	-	O
intersectionOf	intersectionOf1	-	-	O	O
	intersectionOf2	-	-	O	O

	intersectionOf3	-	-	O	O
	intersectionOf4	-	-	O	O

[표 부록 A-1 시스템 제공 룰 파일의 수준]

다) Rule Entailment

Rule Name	if E contains	then add
rdf1_rdfs4a_4b	?p(?x, ?y)	rdf:Property(?p) rdfs:Resource(?x) rdfs:Resource(?y)
rdfs2	rdfs:domain(?p, ?c) ?p(?x, ?y)	?c(?x)
rdfs3	rdfs:range(?p, ?c) ?p(?x, ?y)	?c(?y)
rdfs5	rdfs:subPropertyOf(?a, ?b) rdfs:subPropertyOf(?b, ?c) notEqual(?a, ?b) notEqual(?b, ?c) notEqual(?a, ?c)	rdfs:subPropertyOf(?a, ?c)
rdfs7	rdfs:subPropertyOf(?p, ?q) ?p(?a, ?b)	?q(?a, ?b)

	notEqual(?p, ?q)	
rdfs8	rdfs:Class(?a)	rdfs:subClassOf(?a, rdfs:Resource)
rdfs9	rdfs:subClassOf(?x, ?y) ?x(?a) notEqual(?x, ?y)	?y(?a)
rdfs11	rdfs:subClassOf(?a, ?b) rdfs:subClassOf(?b, ?c) notEqual(?a, ?b) notEqual(?b, ?c) notEqual(?a, ?c)	rdfs:subClassOf(?a, ?c)
rdfs12	rdfs:ContainerMembershipProperty(?x)	rdfs:subPropertyOf(?x, rdfs:member)
rdfs13	rdfs:Datatype(?x)	rdfs:subClassOf(?x, rdfs:Literal)
rdfs_ext1	rdfs:domain(?p, ?x) rdfs:subClassOf(?x, ?y)	rdfs:domain(?p, ?y)
rdfs_ext2	rdfs:range(?p, ?x) rdfs:subClassOf(?x, ?y)	rdfs:range(?p, ?y)
rdfs_ext3	rdfs:domain(?p, ?c) rdfs:subPropertyOf(?x, ?p)	rdfs:domain(?x, ?c)
rdfs_ext4	rdfs:range(?p, ?c)	rdfs:range(?x, ?c)

	rdfs:subPropertyOf(?x, ?p)	
thing1	owl:Class(?c)	rdfs:subClassOf(?c, owl:Thing)
sameAs1	owl:sameAs(?a, ?b) owl:sameAs(?b, ?c) notEqual(?a, ?b) notEqual(?b, ?c)	owl:sameAs(?a, ?c)
sameAs2	owl:sameAs(?x, ?a) owl:sameAs(?y, ?b) ?p(?x, ?y)	?p(?a, ?b)
sameAs3	owl:sameAs(?p, ?q) ?p(?x, ?y) notEqual(?p, ?q) notEqual(?p, owl:sameAs)	?q(?x, ?y)
sameAs4	owl:sameAs(?x, ?y) ?p(?x, ?z) notEqual(?x, ?y) notEqual(?p, owl:sameAs)	?p(?y, ?z)
sameAs5	owl:sameAs(?x, ?y) ?p(?z, ?x) notEqual(?x, ?y) notEqual(?p, owl:sameAs)	?p(?z, ?y)

inverseOf1	owl:inverseOf(?p, ?q) ?p(?a, ?b)	?q(?b, ?a)
inverseOf11	owl:inverseOf(?p, ?q) ?q(?a, ?b)	?p(?b, ?a)
inverseOf2	owl:inverseOf(?p, ?q) rdfs:domain(?p, ?c)	rdfs:range(?q, ?c)
inverseOf3	owl:inverseOf(?p, ?q) rdfs:range(?p, ?c)	rdfs:domain(?q, ?c)
inverseOf4	owl:inverseOf(?p, ?p)	owl:SymmetricProperty(?p)
symmetricProperty 1	owl:SymmetricProperty(?p) ?p(?x, ?y)	?p(?y, ?x)
symmetricProperty 2	owl:SymmetricProperty(?p)	owl:inverseOf(?p, ?p)
functionalProperty1	owl:FunctionalProperty(?p) ?p(?a, ?b) ?p(?a, ?c) notEqual(?b, ?c) notEqual(?p, rdf:type)	owl:sameAs(?b, ?c)
functionalProperty2	owl:FunctionalProperty(?p) owl:inverseOf(?p, ?q)	owl:InverseFunctionalProperty(?q)
inverseFunctional1	owl:InverseFunctionalProperty(?p)	owl:sameAs(?b, ?c)

	?p(?b, ?a) ?p(?c, ?a) notEqual(?b, ?c) notEqual(?p, rdf:type)	
inverseFunctional2	owl:InverseFunctionalProperty(?p) owl:inverseOf(?p, ?q)	owl:FunctionalProperty(?q)
transitiveProperty1	owl:TransitiveProperty(?p) ?p(?a, ?b) ?p(?b, ?c) notEqual(?a, ?b) notEqual(?b, ?c)	?p(?a, ?c)
equivalentClass1	owl:equivalentClass(?x, ?y) notEqual(?x, ?y)	rdfs:subClassOf(?x, ?y) rdfs:subClassOf(?y, ?x)
equivalentClass2	rdfs:subClassOf(?x, ?y) rdfs:subClassOf(?y, ?x) notEqual(?x, ?y)	owl:equivalentClass(?x, ?y)
equivalentProperty 1	owl:equivalentProperty(?p, ?q) notEqual(?p, ?q)	rdfs:subPropertyOf(?p, ?q) rdfs:subPropertyOf(?q, ?p)
equivalentProperty 2	rdfs:subPropertyOf(?p, ?q) rdfs:subPropertyOf(?q, ?p) notEqual(?p, ?q)	owl:equivalentProperty(?p, ?q)

allValuesFrom1	owl:allValuesFrom(?x, ?y) owl:onProperty(?x, ?p) ?x(?a) ?p(?a, ?b)	?y(?b)
allValuesFrom2	owl:onProperty(?r, ?p) owl:allValuesFrom(?r, ?c) owl:onProperty(?s, ?q) owl:allValuesFrom(?s, ?d) rdfs:subPropertyOf(?p, ?q) rdfs:subClassOf(?d, ?c)	rdfs:subClassOf(?s, ?r)
someValuesFrom1	owl:someValuesFrom(?x, ?y) owl:onProperty(?x, ?p) ?y(?q) ?p(?a, ?q)	?x(?a)
someValuesFrom2	owl:onProperty(?r, ?p) owl:someValuesFrom(?r, ?c) owl:onProperty(?s, ?q) owl:someValuesFrom(?s, ?d) rdfs:subPropertyOf(?q, ?p) rdfs:subClassOf(?d, ?c)	rdfs:subClassOf(?s, ?r)
allDifferent1	owl:distinctMembers(?x, ?m)	owl:distinctMembers(?x, ?n)

	rdf:rest(?m, ?n) notEqual(?n, rdf:nil)	
allDifferent2	owl:distinctMembers(?x, ?m) owl:distinctMembers(?x, ?n) rdf:first(?m, ?i) rdf:first(?n, ?j) notEqual(?n, ?m) notEqual(?j, ?i)	owl:differentFrom(?i, ?j)
minCardinality1	owl:onProperty(?r, ?p) owl:minCardinality(?r, "1"^^xsd:nonNegativeInteger) ?p(?i, ?j)	?r(?i)
minCardinality2	owl:onProperty(?r, ?p) owl:someValuesFrom(?r, ?c) owl:onProperty(?s, ?q) owl:minCardinality(?s, "1"^^xsd:nonNegativeInteger) rdfs:subPropertyOf(?p, ?q)	rdfs:subClassOf(?r, ?s)
maxCardinality1	owl:onProperty(?r, ?p) owl:maxCardinality(?r, "1"^^xsd:nonNegativeInteger) ?r(?i)	owl:sameAs(?j, ?k)

	?p(?i, ?j) ?p(?i, ?k) notEqual(?j, ?k)	
oneOf1	owl:oneOf(?c, ?m) rdf:rest(?m, ?n) notEqual(?n, rdf:nil)	owl:oneOf(?m, ?n) rdfs:subClassOf(?m, ?c)
oneOf2	owl:oneOf(?c, ?m) rdf:first(?m, ?i)	?c(?i)
unionOf1	rdf:rest(?x, ?y) owl:unionOf(?c, ?x) notEqual(?y, rdf:nil)	owl:unionOf(?x, ?y) rdfs:subClassOf(?x, ?c)
unionOf2	owl:unionOf(?c, ?x) rdf:first(?x, ?a)	rdfs:subClassOf(?a, ?c)
hasValue1	owl:hasValue(?c, ?y) owl:onProperty(?c, ?p) ?p(?x, ?y)	?c(?x)
hasValue2	owl:hasValue(?x, ?y) owl:onProperty(?x, ?p) ?x(?a)	?p(?a, ?y)
hasValue3	owl:onProperty(?r, ?p) owl:hasValue(?r, ?i)	rdfs:subClassOf(?s, ?r)

	owl:onProperty(?s, ?q) owl:hasValue(?s, ?i) rdfs:subPropertyOf(?q, ?p)	
hasValue4	owl:onProperty(?r, ?p) owl:hasValue(?r, ?i) ?c(?i) owl:onProperty(?s, ?q) owl:someValuesFrom(?s, ?c) rdfs:subPropertyOf(?p, ?q)	rdfs:subClassOf(?r, ?s)
disjointWith1	owl:disjointWith(?c, ?d) ?c(?x) ?d(?y)	owl:differentFrom(?x, ?y)
intersectionOf1	owl:intersectionOf(?c, ?x) rdf:first(?x, ?y)	rdfs:subClassOf(?c, ?x) rdfs:subClassOf(?c, ?y)
intersectionOf2	rdf:rest(?x, ?y) owl:intersectionOf(?c, ?x) notEqual(?y, rdf:nil)	owl:intersectionOf(?x, ?y)
intersectionOf3	rdf:first(?b, ?c) rdf:rest(?b, rdf:nil) owl:intersectionOf(?z, ?b) ?c(?i)	?b(?i)

intersectionOf4	rdf:first(?b, ?c) ?c(?i) ?b(?i) owl:intersectionOf(?n, ?b)	?n(?i)

[표 부록 A-2 Rule Entailment]

라) SWRL Built-Ins

Built-Ins Type	Built-In	Applied
Comparisons	equal	O
	notEqual	O
	lessThan	O
	lessThanOrEqual	O
	greaterThan	O
	greaterThanOrEqual	O
Math	add	O
	subtract	O
	multiply	O
	divide	O
	integerDivide	O
	mod	O

	pow	O
	unaryPlus	O
	unaryMinus	O
	abs	O
	ceiling	O
	floor	O
	round	O
	roundHalfToEven	O
	sin	O
	cos	O
	tan	O
Boolean Values	booleanNot	O
Strings	stringEqualIgnoreCase	O
	stringConcat	O
	substring	O
	stringLength	O
	normalizeSpace	O
	upperCase	O
	lowerCase	O
	translate	O

	contains	O
	containsIgnoreCase	O
	startsWith	O
	endsWith	O
	substringBefore	O
	substringAfter	O
	matches	O
	replace	O
	tokenize	O
Date, Time and Durations	yearMonthDuration	O
	dayTimeDuration	O
	dateTime	O
	date	O
	time	O
	addYearMonthDurations	O
	subtractYearMonthDurations	O
	multiplyYearMonthDuration	O
	divideYearMonthDurations	O
	addDayTimeDurations	O
	subtractDayTimeDurations	O

	multiplyDayTimeDurations	O
	divideDayTimeDuration	O
	subtractDates	O
	subtractTimes	O
	addYearMonthDurationToDateTime	O
	addDayTimeDurationToDateTime	O
	subtractYearMonthDurationFromDateTime	O
	subtractDayTimeDurationFromDateTime	O
	addYearMonthDurationToDate	O
	addDayTimeDurationToDate	O
	subtractYearMonthDurationFromDate	O
	subtractDayTimeDurationFromDate	O
	addDayTimeDurationToTime	O
	subtractDayTimeDurationFromTime	O
	subtractDateTimesYieldingYearMonthDuration	O
	subtractDateTimesYieldingDayTimeDuration	O
URIs	resolveURI	O
	anyURI	O
Lists	listConcat	-
	listIntersection	-

	listSubtraction	-
	member	-
	length	-
	first	-
	rest	-
	sublist	-
	empty	-

[표 부록 A-3 SWRL Built-Ins]

B. 색 인

ㅅ

서버의 시작 11

서버의 중지 12

스토어의 삭제 34

스토어의 생성 33

ㅋ

클라이언트 API 70

A

Ask 쿼리 37

C

Construct 쿼리 38

D

Describe 쿼리 38

F

File Full Build 35

File Increment Build 35

I

Index Service 27

Internal Database 25

M

Manual Add Build 36

Manual Delete Build 37

Q

Query Processor 28

R

Repository Service 27

S

Select 쿼리 37

Server Manager 25

ServerBuildManager.java 73

SparqlQuery.java 74

StoreManager.java 72

T

Triple Store 26