Java에 기반한 DB 프로그래밍

지난 호에서는 DB에서의 자바 접근 지원 방안에 대해서 살펴 보았다. 이번 호에서는 실제 프로그래밍 사례로서 오라클8i 상에서의 자바 기반 DB 프로그래밍 방안에 대해서 살펴 보기로 한다. JDBC에 기반한 프로그래밍 방안은 이미 많이 알려져 있으므로 이번 호에서는 Java 저장 프로시저의 작성 방법 및 SQLJ에 기반한 프로그래밍 방안에 대해서 기술하기로 한다. CORBA 및 EJB에 기반한 프로그래밍 방법은 다음 호에서 다루기로 한다.

■ 장성우/ 한국 오라클 제품기획실 서버기술팀장
(swchang@kr.oracle.com)

연 재 순 서

1. Java와 DB의 통합
2. 오라클8i 상에서의 Java 활용
3. CORBA 및 EJB에 기반한 프로그래밍 방법

1. 자바 저장 프로시저 작성 방법

자바 저장 프로시저는 데이터베이스 안에 저장되어 SQL 및 다른 응용 프로그램에 의해 호출되어 사용될 수 있는 자바 프로그램을 나타낸다. 일반적으로 그래픽 사용자 인터페이스를 제외한 어떠한 자바 프로그램도 저장 프로시저로서 데이터베이스 내에 저장되고 사용될 수 있다.

자바 저장 프로시저의 작성 방법은 다음과 같다.

▲ 자바 소스의 작성
▲ 자바 소스를 컴파일
▲ 컴파일된 자바 클래스를 데이터베이스 안으로 적재
▲ 자바 클래스에 PL/SQL 인터페이스를 씌움
▲ PL/SQL 인터페이스를 통한 자바 클래스 호출

1.1 자바 소스의 작성

자바 저장 프로시저를 작성하기 위한 첫 단계로서 당연히 먼저 자바 프로그램 소스를 작성한다. 예를 들어 다음과 같은 예제 프로그램을 작성했다고 가정해 보자.

```
Public class sample {
    public static String hello() {
            return "Hello";
    }
}
```

1.2 자바 소스를 컴파일

자바 소스의 작성이 끝나면 해당 소스를 자바 컴파일러를 통해 컴파일 해야 한다.

```
Javac sample.java
```

1.3 자바 클래스 파일을 데이터베이스로 적재

컴파일이 성공적으로 수행되면 컴파일 결과로 생성된 자바 클래스 파일을 데이터베이스에 적재한다. 이를 위해 'loadjava'라는 명령을 사용한다.

```
loadjava -u scott/tiger sample.class
```

이 명령을 통해 데이터베이스의 사용자 'scott'의 스키마 안에 jdbc1.class가 자바 서버 객체로 등록되어지게 된다.

1.4 자바 클래스를 위한 PL/SQL 인터페이스 생성

이제 자바 클래스가 데이터베이스 안에 저장되었다. 하지만 자바 메소드를 SQL로부터 호출하기 전에 이 자바 프로그램의 존재를 데이터베이스 안에 등록시켜서 다른 프로그램들이 자바 프로그램을 호출할 수 있는 길을 터 주어야 한다. 이를 위해 호출 명세(call specification)라고 불리우는 PL/SQL 인터페이스를 자바 프로그램마다 정의해 주어야 한다. 앞서 등록된 자바 프로그램을 위한 호출 명세를 작성하면 다음과 같다.

```
create or replace function java_sample
return varchar2
as language java
name 'sample.hello() return java.lang.String';
```

이 호출 명세는 sample 자바 클래스 안에 있는 hello() 함수가 PL/SQL 함수 java_sample()로 호출될 수 있도록 해 준다. 이 호출 명세를 통해 등록된 자바 메소드는 'java_sample'이라는 이름으로 다른 SQL 및 PL/SQL 루틴에 의해 사용되어질 수 있게 된다.

1.5 SQL 및 PL/SQL 루틴에서의 자바 프로그램 호출

자바 프로그램을 위한 호출 명세의 작성이 완료되면 이제 해당 자바 프로그램을 다른 SQL 및 PL/SQL 프로그램에서 호출하여 사용할 수 있게 된다.

간단히 SQL에서 호출하는 방법은 다음과 같다.

```
Select java_sample()
From dual;
```

또, PL/SQL 루틴에서는 함수 호출의 방법으로 호출하여 사용할 수 있다.

```
myString varchar2;
call java_sample() into :myString;
print myString;
```

2. SQLJ 프로그래밍 방안

2.1 SQLJ 개요

자바 프로그램 안에서 데이터베이스에 접속하기 위해 일반적으로는 JDBC 호출을 사용한다. 하지만, JDBC를 통한 데이터베이스 접속 절차 없이 자바 프로그램 안에서 곧바로 SQL 문장을 호출하여 사용할 수 있으면 프로그램을 훨씬 쉽게 작성할 수 있을 것이다. 이와 같이 자바 프로그램 안에 JDBC 루틴을 사용하지 않고 SQL을 곧 바로 내장(embed)시킨 것을 SQLJ 프로그램이라고 한다.

생성된 SQLJ 프로그램은 SQLJ 전처리기(preprocessor)에 의해 JDBC 호출을 포함하는 자바 코드로 변환되어지게 된다. 그러면, 이제 변환된 자바 코드를 이용하여 자바 컴파일러를 통해 통상적인 자바 프로그램 수행 절차를 밟으면 된다.

그러면, 왜 SQLJ를 사용하는가? 무엇보다도 JDBC를 사용하는 것보다 프로그램을 개발하고 관리하기 쉽다는 점을 들 수 있다. 다음의 두 프로그램을 비교해 보자.

```
// SQLJ
int n = 17950;
#sql {insert into EMP values (:n)};

// JDBC
int n = 17950;
Statement stmt =
 conn.prepareStatement
  ("insert into EMP values (?)");
stmt.setInt (1, n);
stmt.execute ();
stmt.close ();
```

이 밖에도 다음과 같은 장점을 얻을 수 있다.

▲ SQL의 문맥 에러(syntax error) 및 자바와 SQL 사이의 타입 불일치(type mismatch)를 전처리 단계에서 감지할 수 있다.

▲ SQL 선행 컴파일(pre-compilation) 기법을 이용하여 빠른 실시간 수행 속도를 얻는다.

▲ DB와 관련된 자바 코드는 대부분 SQL의 처리가 핵심적인데 이 부분에 대한 처리가 SQL의 지식으로도 충분히 가능하게 된다.

2.2 SQLJ 프로그램 작성

SQLJ 프로그램의 기본적인 구조는 다음과 같다.

```
import sqlj.runtime.*;    // SQLJ runtime을 사용
import sqlj.runtime.ref.*; // 또 다른 SQLJ runtime
import java.sql.*;        // JDBC 역시 사용하기 때문
class ExampleTemplate {
  public static void main (String args [])
    throws SQLException {
      .....
      #sql{
        // SQL 문장이 이 곳에 위치하게 됨
      }
      .....
    }
}
```

SQLJ 프로그램 작성 후의 작업 순서는 다음과 같다.

● SQLJ 프로그램의 전처리를 수행한다(SQLJ 프로그램은 확장자가 '.sqlj'가 되도록 한다)

```
sqlj Example.sqlj
```

● 전처리를 통해 생성된 자바 소스를 컴파일한다

```
javac Example.java
```

● 생성된 자바 클래스를 수행한다

```
java Example
```

또는 앞서 설명한 바와 같이 자바 저장 프로시저로 등록하여도 된다.

SQLJ를 사용하기 위해 필요한 환경 설정은 다음과 같다.

● CLASSPATH에 $ORACLE_HOME/sqlj/lib/translator.zip 추가

● PATH에 $ORACLE_HOME/sqlj/bin 추가

또, SQLJ 프로그램 안에 다음의 루틴이 추가되어야 한다.

```
import sqlj.runtime.*;
import java.sql.*;        // 결국은 JDBC 호출을 사용하기 때문이다
```

3. 요약

이번 호에서는 자바에 기반하여 자바 저장 프로시저를 만드는 방법과 데이터베이스 접근시 SQLJ를 활용하는 방안에 대하여 설명하였다. 다음 호에서는 데이터베이스 상에서 EJB 및 CORBA를 활용하여 자바 프로그램을 작성하는 방법에 대하여 설명하기로 한다.