오라클8i 상에서의 프로그래밍 방법

지난호에서는 실제 프로그래밍 사례로서 오라클8i 상에서의 자바 기반 DB 프로그래밍 방안 중에서 자바 저장 프로시져 및 SQLJ 프로그래밍 방안에 대해서 살펴 보았다. 이번 호에서는 또 다른 자바 기반 DB 프로그래밍 방법인 CORBA 및 EJB 프로그밍에 대해서 살펴 보기로 한다. 다만 지면 관계로 CORBA와 EJB에 대해서 깊이 다룰 수 없으므로 각각에 대한 기본적인 소개만을 한 후, 곧바로 오라클8i 상에서의 프로그래밍 방법에 대해서 다루도록 한다. 예제로서는 CORBA와 EJB 모두 숫자값을 증감시켜 주는 카운터(Counter) 객체 프로그램을 작성해 보도록 한다.

■ 장성우/한국오라클 제품기획실 서버기술팀
(Email:swchang@kr.oracle.com)

연재순서

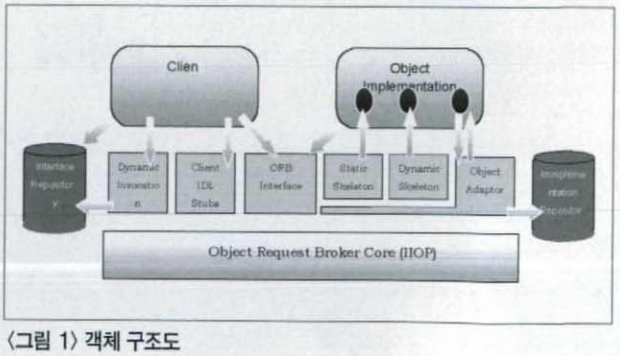
1. Java와 DB의 통합
2. 오라클8i 상에서의 Java 활용
3. CORBA 및 EJB에 기반한 프로그래밍 방법

1. CORBA 프로그램 작성 방법

1.1 CORBA 개요

CORBA는 OMG(Object Management Group)에 의해 정의된 분산 객체 환경의 구축 표준안이다. 현재 버전은 2.3까지 나와 있으며 그 구조는 〈그림 1〉과 같다.

- 클라이언트 IDL Stubs : 객체에 대한 정적인 인터페이스를 제공한다.
- 동적 호출 인터페이스(Dynamic Invocation Interface:DII) : 실행 시에 호출할 메소드를 찾는 것을 도와준다.
- 인터페이스 저장소(Interface Repository) : 등록된 모든 객체들의 메소드, 인자들 및 요약 설명 등을 변경하고 얻을 수 있게 하는 정보 저장소
- ORB 인터페이스 : 응용 프로그램에 적용될 수 있는 지역 서비스(local services)에 대한 몇 개의 API로 구성되어 있다.



〈그림 1〉 객체 구조도

● 서버 IDL Stubs(Skeleton) : 서버에 의해서 수행될 각각의 객체 기능에 대한 정적인 인터페이스를 제공한다.

● 동적 뼈대 인터페이스(Dynamic Skeleton Interface :DSI) : skeleton을 가지고 있지 않은 객체들의 메소드를 호출할 수 있도록 실행시 연결(runtime binding)을 제공한다.

● 객체 어댑터(Object Adapter) : ORB의 기본 통신 서비스의 맨 윗부분에 위치하며, 서버의 객체대신에 service의 request를 받는다. 서버 객체들의 인스턴스를 생성하고, 객체 IDs를 assign하고, request를 넘기는 runtime 환경을 제공한다.

● 구현 저장소(Implementation Repository) : CORBA 객체들의 인스턴스, ID등 서버가 지원하는 클래스들에 대한 실행시 정보 저장소를 제공

1.2 오라클8i 상에서의 CORBA 프로그램 작성 방법

오라클8i의 ORB는 Visibroker를 이용하기 때문에 크게 기존의 CORBA 객체들을 위한 응용 프로그램을 그대로 이용할 수가 있으나 단지, 객체 어댑터에 등록하는 절차 등은 필요가 없다.

1) IDL 파일 생성

먼저 IDL 파일을 생성한다. IDL(Interface Definition Language) 파일은 클라이언트와 서버가 공통적으로 접근할 객체를 특정 프로그래밍 언어에 종속되지 않고 독립적으로 표현할 수 있도록 지원한다.

```
// count.idl
module Counter {
  interface Count
  { attribute long sum;
    long increment();
  };
};
```

2) IDL 파일로부터 stub 및 skeleton 생성

IDL은 프로그래밍 언어가 아니므로 우리가 사용할 언어(여기에서는 자바)로 변환되어야 한다. 이 작업은 idl2java 프로그램을 이용해서 IDL 파일로부터 연관된 stub 및 skeleton 자바 파일들을 생성한다. 이 때 "java.lang.ClassCastException ..."과 같은 에러가 발생할 수 있는 데, 이것은 필요한 jar파일들이 classpath에 잡혀있지 않은 경우이다.

```
idl2java -no_comments -no_tie count.idl
```

3) 구현 파일 수정

컴파일 결과로 생성된 skeleton 구현 파일(_example_count.java파일)을 CountImpl.java로 이름을 변경한 후 다음과 같이 수정한다.

프로그램에서 굵은 글씨체로 나타나는 부분이 Oracle 8i를 위한 코딩이다.

```
// CountImpl.java
package Counter;
public class CountImpl extends Counter._CountImplBase
          implements oracle.aurora.AuroraServices.ActivatableObject {
  private int sum;

  public CountImpl(java.lang.String name) {
    super(name);
  }
  public org.omg.CORBA.Object _initializeAuroraObject() {
    return this;
  }
  public CountImpl() {
    super();
  }
  public int increment() {
    this.sum++;
    return this.sum;
  }
  public void sum(int sum) {
    this.sum = sum;
  }
  public int sum() {
    return this.sum;
  }
}
```

4) 모든 java 파일을 컴파일

IDL 파일의 전환 결과 생성된 자바 파일들과 앞의 구현 파일을 모두 컴파일한다. 이때 앞에서 추가된 부분의 인터페이스를 위해서 aurora_client.jar를 classpath에 추가하고 난 후 다음과 같이 수행한다.

```
javac Counter\*.java
```

5) JAR 파일 생성

CORBA 객체들을 오라클8i에 적재하기 위해서 컴파일된 클래

스 파일들을 jar로 묶는 데 다음과 같이 수행한다.

```
jar cf0 ServerObjects.jar Counter\*.class
```

6) JAR 파일을 데이터베이스로 적재

앞에서 생성한 ServerObjects.jar파일을 데이터베이스 서버에 적재하기 위해서 loadjava 명령어를 수행한다.

```
loadjava -u scott/tiger -oci8 -verbose -resolve ServerObjects.jar
```

7) Publish 수행

파일 적재를 끝낸 후에 publishing을 해야 한다. 이것은 한마디로 CORBA 객체들을 명칭 서비스에 등록하는 것인 데 다음의 경우 "test"라는 root context 밑에 MyUsername이라는 이름으로 CORBA 객체들을 등록하고 있다.

```
publish -user scott -password tiger -service sess_iiop://krservers1:2222
/test/CountObj
Counter.CountImpl Counter.CountHelper -schema SCOTT
```

8) 클라이언트 프로그램을 작성

마지막으로 클라이언트 프로그램을 작성한다. 클라이언트 프로그램은 내부적으로 다음과 같은 루틴을 가진다.

① JNDI(Java Naming and Directory Interface) context 환경을 설정한다.
② 초기 참조를 얻기 위해 JNDI를 사용한다.
③ publish되어진 서버 객체들을 찾아서(lookup) 활성화(activate)시킨다.
④ 서버 객체들의 increment( ) 메소드를 실행한다.

```
// CountClient.java 파일
import java.net.*;
import java.util.*;
import javax.naming.*;
import Counter.*;
import oracle.aurora.jndi.sess_iiop.ServiceCtx;

class CountClient
{
 public static void main(String args[]) {
  try {
   Count counter = null;
   String CountURL = args.length > 0 ? args[0] :
    "sess_iiop://krservers1:2222/test/CountObj";
   String username = args.length > 1 ? args[1] : "scott";
   String password = args.length > 2 ? args[2] : "tiger";
   //Use a credential login, See the Authentication section for more info.
   Hashtable env = new Hashtable();
            env.put(javax.naming.Context.URL_PKG_PREFIXES,
"oracle.aurora.jndi");
   env.put(javax.naming.Context.SECURITY_PRINCIPAL, username);
   env.put(javax.naming.Context.SECURITY_CREDENTIALS, password);
env.put(javax.naming.Context.SECURITY_AUTHENTICATION,
ServiceCtx.NON_SSL_LOGIN);
   try {
    Context ic = new InitialContext(env);
    counter = (Counter.Count)ic.lookup(CountURL);
   }
   catch(CommunicationException exc) {
    System.out.println("Unable to connect using " + CountURL);
    System.exit(1);
   }
   catch(oracle.aurora.jndi.sess_iiop.ActivationException exc) {
    System.out.println("Activation error: " + exc.getMessage());
    System.exit(1);
   }
   catch(NamingException exc) {
    System.out.println("Invalid URL: " + CountURL);
    System.exit(1);
   }

   counter.sum((int)0);

   // Calculate Start time
   long startTime = System.currentTimeMillis();

   // Increment 1000 times
   System.out.println("Incrementing");
   for (int i = 0 ; i < 1000 ; i++ )
   { counter.increment();
   }

   // Calculate stop time; print out statistics
   long stopTime = System.currentTimeMillis();
   System.out.println("Avg Ping = "
            + ((stopTime - startTime)/1000f) + " secs");
   System.out.println("Sum = " + counter.sum());
  } catch(Exception e)
  { System.err.println("System Exception");
   System.err.println(e);
  }
 }

}
```

9) 클라이언트 프로그램의 컴파일 및 실행

앞의 CountClient.java를 컴파일한 후 실행을 한다. 이 때, 몇

가지 주의할 점들이 있다.

① classpath에 다음의 jar가 추가되어야 한다. "aurora_client.jar, vbjorb.jar, vbjapp.jar, vbjtools.jar:

② 앞의 jar파일들은 Visibroker v3.2 이상 버전을 이용해야 한다.

③ CORBA 객체들을 publishing한 후 데이터베이스 서버를 재시동(restart) 시킬 필요는 없다.

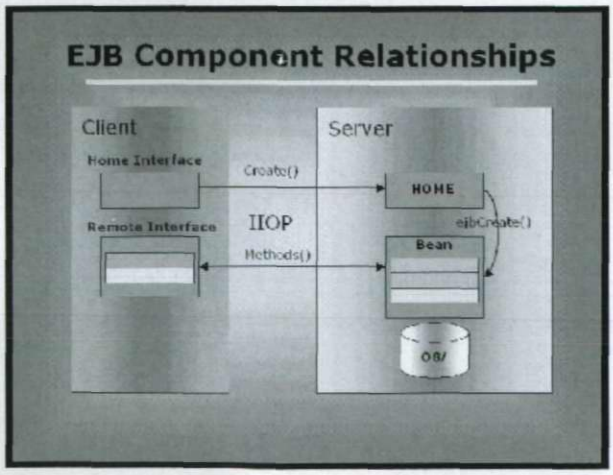
2. EJB 프로그래밍 방안

EJB는 다단계 네트워크 환경에서 분산 객체 구조에 기반한 자바 응용프로그램의 개발 및 확장을 위한 컴포넌트 모델이다. 여기서 컴퍼넌트 모델이란 재사용 가능한 응용 프로그램 요소의 작성을 지원하기 위한 프로그래밍 방안이며, 컴포넌트란 개발하는 프로그램에 추가될 수 있는 프로그램의 기정의 모듈이라고 볼 수 있다. 컴포넌트 모델의 예로는 JavaBeans를 들 수 있고, Enterprise JavaBeans는 서버 상에서 동작하는 서버 프로그램 요소들의 작성을 지원하기 위한 JavaBeans 컴포넌트 모델의 확장형이라 할 수 있다.

EJB는 다음과 같은 4 개 중요 구성 요소들로 구성되어 있다.

- 홈 인터페이스(home interface)
- 원격지 인터페이스(remote interface)
- Bean(remote interface)의 구현(implementation)
- 전개 설명자(deployment descriptor)

이들 구성 요소간의 관계는 〈그림 2〉와 같다.



〈그림 2〉 EJB는 구성 요소 관계

홈 인터페이스는 Container가 자동적으로 구현하는 홈 객체의 인터페이스이다. 그러므로, 홈 객체의 구현은 필요 없다. 홈은 EJB의 생성자(factory) 객체이다. 원격지 인터페이스는 Bean에 구현한 메소드를 기술한다.

이 메소드가 우리가 사용할 응용 프로그램 로직을 나타내게 된다. Bean 구현은 원격지 인터페이스의 구현 코드와 컨테이너에서 요구하는 메소드를 구현한다. 전개 설명자는 Bean의 속성을 기술한다.

현재 EJB의 종류에는 세션 빈과 엔터티 빈이 있는데 오라클8i는 엔터티 빈만 지원한다.

2.1 오라클8i 상에서의 EJB 프로그램 작성 방법

1) 홈 인터페이스 파일을 생성한다.

```
// EjbCountHome.java 파일
package ejb.oracle8i;
import javax.ejb.*;
import java.rmi.RemoteException;

public interface EjbCountHome extends EJBHome {
        public EjbCount create() throws RemoteException;
}
```

2) 원격지 인터페이스 파일을 생성한다.

```
// EjbCount.java 파일
package ejb.oracle8i;
public interface EjbCount extends javax.ejb.EJBObject
{
  public void increment() throws java.rmi.RemoteException;
  public void setSum(int a) throws java.rmi.RemoteException;
  public int getSum() throws java.rmi.RemoteException;
}
```

3) 원격지 인터페이스를 구현한 구현 파일을 작성한다.

```
// EjbCountImpl.java 파일
package ejb.oracle8i;
import javax.ejb.*;
import java.util.Properties;
import javax.jts.util.*;
```

```
import org.omg.CosTransactions.*;
public class EjbCountImpl implements javax.ejb.SessionBean {
 private int sum;
 private SessionContext ctx = null;
 public void ejbCreate() {
 }
 public void setSessionContext(SessionContext ctxArg) {
  ctx = ctxArg;
 }
 public void ejbActivate() { }
 public void ejbPassivate() { }
 public void ejbRemove() { }
 public void increment() {
  this.sum++;
 }
 public void setSum(int a) {
  this.sum = a;
 }
 public int getSum() {
  return this.sum;
 }
}
```

4) 지금까지 생성한 파일들을 컴파일 한다.

```
javac -d . *.java
```

5) 전개 설명자 파일을 생성한다.

```
//EjbCount.ejb
SessionBean ejb.oracle8i.EjbCountImpl {

 // This is the published name of the bean
 BeanHomeName = "test/EjbCountObj";
 RemoteInterfaceClassName = ejb.oracle8i.EjbCount;
 HomeInterfaceClassName = ejb.oracle8i.EjbCountHome;

 // add exact semantics for beta
 AllowedIdentities = { SCOTT };
 RunAsMode = CLIENT_IDENTITY;

 TransactionAttribute = TX_SUPPORTS;

  // Add any environment properties that the bean requires (need read
example)
 EnvironmentProperties {
  prop1 = value1;
  prop2 = "value two";
 }

 public void increment() {
  RunAsMode = CLIENT_IDENTITY;
  AllowedIdentities = { SCOTT };
 }

 public void setSum(int a) {
  RunAsMode = CLIENT_IDENTITY;
  AllowedIdentities = { SCOTT };
 }

 public int getSum() {
  RunAsMode = CLIENT_IDENTITY;
  AllowedIdentities = { SCOTT };
 }
}
```

6) 지금까지 생성된 모든 자바 클래스 파일들 및 전개 설명자 파일을 jar화일로 만든다.

```
jar cf0 Counter.jar ejb
```

7) 서버에서 loadjava 명령어를 이용해서 jar 파일에 들어있는 EJB 객체들을 데이터베이스 서버에 적재한다.

```
loadjava -u scott/tiger -oci8 -verbose -resolve Counter.jar
```

8) 오라클8i의 Naming service에 등록을 한다.

```
Publish -user scott -password tiger -service sess_iiop://krservers1:2222
/test/CountObj
Counter.CountImpl Counter.CountHelper -schema scott
```

9) 클라이언트 프로그램을 작성한다.

```
//EjbCountClient.java 파일
import java.io.*;
import java.sql.*;
import java.util.*;
import javax.naming.*;
import ejb.oracle8i.*;
import java.rmi.RemoteException;
import oracle.aurora.jndi.sess_iiop.*;

class EjbCountClient {
 private EjbCount counter;
```

```
public static void main (String[] args) throws java.io.IOException {
CountClient a = new CountClient();
 String ejb_test_url = args.length > 0 ? args[0] : "sess_iiop://
krservers1:2222";
String username = args.length > 1 ? args[1] : "scott";
String password = args.length > 2 ? args[2] : "tiger";
String beanName = "/test/dbwork";
Hashtable environment = new Hashtable();
 environment.put(javax.naming.Context.URL_PKG_PREFIXES,
"oracle.aurora.jndi");
environment.put(Context.SECURITY_PRINCIPAL, username);
environment.put(Context.SECURITY_CREDENTIALS, password);
      environment.put(Context.SECURITY_AUTHENTICATION,
ServiceCtx.NON_SSL_LOGIN);

EjbCountHome home = null;
try {
Context ic = new InitialContext(environment);
System.out.println("step1");
//AuroraTransactionService.initialize(ic, ejb_test_url);
System.out.println("step2");
//home = (DatabaseWorkHome)ic.lookup(ejb_test_url);
home = (DatabaseWorkHome)ic.lookup(ejb_test_url + beanName);
System.out.println("step3");
}
catch (ActivationException e) {
System.out.println("Unable to activate : " + e.getMessage());
System.exit(1);
}
catch (CommunicationException e) {
System.out.println("Unable to connect: " + ejb_test_url);
System.out.println(e);
System.exit(1);
}
catch (NamingException e) {
 System.out.println("Invalid URL: " + ejb_test_url);
 System.out.println(e);
 System.exit(1);
}
catch (Exception e) {
 System.out.println("I can't understand what the problem is");
}
counter = home.create();
System.out.println("Setting sum to 0");
a.counter.setSum((int)0);
// Calculate Start time
long startTime = System.currentTimeMillis();

// Increment 1000 times
System.out.println("Incrementing");
for (int i = 0 ; i < 1000 ; i++ )
{ a.counter.increment();
}

// Calculate stop time; print out statistics
long stopTime = System.currentTimeMillis();
System.out.println("Avg Ping = "
            + ((stopTime - startTime)/1000f) + " secs");
System.out.println("Sum = " + a.counter.getSum());
}
}
```

10) 클라이언트 프로그램을 컴파일하고 실행시킨다.

```
Javac CountClient.java
java  CountClient
```