Tibero

External Procedure 안내서

Tibero 7

대한민국 경기도 성남시 분당구 황새울로258번길 29, BS 타워 9층 우)13595

Website

http://www.tmaxtibero.com

기술서비스센터

Tel : +82-1544-8629

E-Mail : info@tmax.co.kr

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses. : OpenSSL, RSA Data Security, Inc., Apache Foundation, Jean-loup Gailly and Mark Adler, Paul Hsieh's hash

Detailed Information related to the license can be found in the following directory : ${INSTALL_PATH}/license/oss_licenses

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다. : OpenSSL, RSA Data Security, Inc., Apache Foundation, Jean-loup Gailly and Mark Adler, Paul Hsieh's hash

관련 상세한 정보는 제품의 다음의 디렉터리에 기재된 사항을 참고해 주십시오. : ${INSTALL_PATH}/license/oss_licenses

안내서 정보

안내서 제목: Tibero External Procedure 안내서

발행일: 2024-08-22

소프트웨어 버전: Tibero 7.2.2

안내서 버전: v7.2.2

내용 목차

그림 목차

안내서에 대하여

안내서의 대상

본 안내서는 Tibero®(이하 Tibero)에서 제공하는 External Procedure의 기본적인 개념과 이를 생성하고 사용하는 방법을 알고자 하는 애플리케이션 프로그램 개발자를 대상으로 기술한다.

안내서의 전제 조건

본 안내서를 원활히 이해하기 위해서는 다음과 같은 사항을 미리 알고 있어야 한다.

- 데이터베이스의 이해
- RDBMS의 이해
- SQL의 이해
- tbPSM의 이해

안내서의 제한 조건

본 안내서는 Tibero를 실무에 적용하거나 운용하는 데 필요한 모든 사항을 포함하고 있지 않다. 따라서 설치, 환경설정 등 운용 및 관리에 대해서는 각 제품 안내서를 참고하기 바란다.

참고

Tibero의 설치 및 환경설정에 관한 내용은 "Tibero 설치 안내서"를 참고한다.

안내서 구성

Tibero External Procedure 안내서는 총 8개의 장으로 구성되어 있다.

각 장의 주요 내용은 다음과 같다.

- 제1장: External Procedure 소개

  External Procedure의 기본 개념과 생성 유형을 간략히 소개한다.

- 제2장: C External Procedure

  C External Procedure의 기본 개념과 주요 특징, 동작 과정을 기술한다.

- 제3장: C External Procedure 생성

  C External Procedure를 생성하기 위한 기본 환경을 설정하는 방법과 C External Procedure의 생성 절차를 기술한다.

- 제4장: C External Procedure 사용

  C External Procedure에서 PSM 파라미터와 C 파라미터를 매핑하는 방법과 Callback Service를 사용하는 방법을 기술한다.

- 제5장: C External Procedure 유틸리티

  사용자 공유 라이브러리 함수를 작성하는 데 유용한 C External Procedure 유틸리티를 기술한다.

- 제6장: Java External Procedure

  Java External Procedure의 기본 개념과 주요 특징, 동작 과정을 기술한다.

- 제7장: Java External Procedure 생성

  Java External Procedure를 생성하기 위한 기본 환경을 설정하는 방법과 Java External Procedure의 생성 절차를 기술한다.

- 제8장: Java External Procedure 사용

  Java External Procedure를 사용하여 기본적인 Java 애플리케이션 프로그램을 개발하는 방법과 Internal JDBC Driver를 사용하는 방법을 기술한다.

안내서 규약

표기	의미
<<AaBbCc123>>	프로그램 소스 코드의 파일명
<Ctrl>+C	Ctrl과 C를 동시에 누름
[Button]	GUI의 버튼 또는 메뉴 이름
진하게	강조
" "(따옴표)	다른 관련 안내서 또는 안내서 내의 다른 장 및 절 언급
'입력항목'	화면 UI에서 입력 항목에 대한 설명
하이퍼링크	메일 계정, 웹 사이트
>	메뉴의 진행 순서
+----	하위 디렉터리 또는 파일 있음
|----	하위 디렉터리 또는 파일 없음
참고	참고 또는 주의사항
주의	주의할 사항
[그림 1.1]	그림 이름
[예 1.1]	예제 이름
`AaBbCc123`	Java 코드, XML 문서
[*command argument*]	옵션 파라미터
< xyz >	'<'와 '>' 사이의 내용이 실제 값으로 변경됨
|	선택 사항. 예) A|B: A나 B 중 하나
…	파라미터 등이 반복되어서 나옴
${ }	환경변수

시스템 사용 환경

	요구 사항
Platform	HP-UX 11i v3(11.31)
	Solaris (Solaris 11)
	AIX (AIX 7.1/AIX 7.2/AIX 7.3)
	GNU (X86, 64, IA64)
	Red Hat Enterprise Linux 7 kernel 3.10.0 이상
	Windows(x86) 64bit
Hardware	최소 2.5GB 하드디스크 공간
	1GB 이상 메모리 공간
Compiler	PSM (C99 지원 필요)
	tbESQL/C (C99 지원 필요)

관련 안내서

안내서	설명
Tibero 설치 안내서	설치 시 필요한 시스템 요구사항과 설치 및 제거 방법을 기술한 안내서이다.
Tibero tbCLI 안내서	Call Level Interface인 tbCLI의 개념과 구성요소, 프로그램 구조를 소개하고 tbCLI 프로그램을 작성하는 데 필요한 데이터 타입, 함수, 에러 메시지를 기술한 안내서이다.
Tibero 애플리케이션 개발자 안내서	각종 애플리케이션 라이브러리를 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero JDBC 개발자 안내서	Tibero에서 제공하는 JDBC 기능을 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero tbESQL/C 안내서	C 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero tbESQL/COBOL 안내서	COBOL 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero tbPSM 안내서	저장 프러시저 모듈인 tbPSM의 개념과 문법, 구성요소를 소개하고, 프로그램을 작성하는 데 필요한 제어 구조, 복합 타입, 서브 프로그램, 패키지와 SQL 문장을 실행하고 에러를 처리하는 방법을 기술한 안내서이다.
Tibero tbPSM 참조 안내서	저장 프러시저 모듈인 tbPSM의 패키지를 소개하고, 이러한 패키지에 포함된 각 프러시저와 함수의 프로토타입, 파라미터, 예제 등을 기술한 참조 안내서이다.
Tibero 관리자 안내서	Tibero의 동작과 주요 기능의 원활한 수행을 보장하기 위해 DBA가 알아야 할 관리 방법을 논리적 또는 물리적 측면에서 설명하고, 관리를 지원하는 각종 도구를 기술한 안내서이다.
Tibero 유틸리티 안내서	데이터베이스와 관련된 작업을 수행하기 위해 필요한 유틸리티의 설치 및 환경설정, 사용 방법을 기술한 안내서이다.
Tibero TAS 안내서	Tibero Active Storage(TAS)를 사용해서 Tibero의 파일을 관리하고자 하는 관리자를 대상으로 기술한 안내서이다.
Tibero	Tibero를 사용하는 도중에 발생할 수 있는 각종 에러의 원인과 해결 방법을 기술한 안내서이다.

안내서	설명
에러 참조 안내서	
Tibero 참조 안내서	Tibero의 동작과 사용에 필요한 초기화 파라미터와 데이터 사전, 정적 뷰, 동적 뷰를 기술한 참조 안내서이다.
Tibero SQL 참조 안내서	데이터베이스 작업을 수행하거나 애플리케이션 프로그램을 작성할 때 필요한 SQL 문장을 기술한 참조 안내서이다.
Tibero Spatial 참조 안내서	Tibero에서 Geometry 타입에 대한 설명과 Spatial 기능 관련 프러시저 함수 목록 및 사용 방법 등을 기술한 안내서이다.
Tibero TEXT 참조 안내서	Tibero의 제공하는 Text Index를 소개하고, Text Index를 생성 하고 사용하는 방법을 기술하는 안내서이다.
Tibero TDP.NET 안내서	Tibero Data Provider for .NET 기능을 기술하는 안내서이다.
Tibero IMCS 안내서	Tibero에서 제공하는 In-Memory Column Store(이하 IMCS) 기능을 기술하는 안내서이다.

제1장 External Procedure 소개

본 장에서는 Tibero에서 제공하는 External Procedure를 간략히 소개한다.

1.1. 개요

External Procedure는 DBMS의 SQL이나 PSM 등으로는 구현하기 어렵거나 구현할 수 없는 것을 C나 Java 프로그래밍 언어로 개발하여 이것을 마치 PSM 함수(또는 프러시저)인 것처럼 DBMS 내에서 사용할 수 있는 기능이다.

사용자는 이 기능을 이용하여 직접 작성한 공유 라이브러리를 필요한 순간에 DBMS 내에서 사용할 수 있다. 그뿐만 아니라 처리 결과까지 전달받을 수 있다.

예를 들면 External Procedure는 다음과 같은 목적에 사용할 수 있다.

- DBMS 내에 특정 이벤트가 발생한 경우 이메일, SMS, 네트워크 등으로 알릴 때
- 디스크 전체나 특정 파일의 크기에 따라 특정 파일을 압축하거나 외부 디스크로 이동시키고 싶을 때
- 데이터에 암호화를 적용하고 싶을 때

1.2. 생성 유형

Tibero에서는 C나 Java 프로그래밍 언어를 이용하여 External Procedure를 생성할 수 있다.

- C External Procedure
  - 동작 과정
  - 생성 절차
  - 사용 예제
- Java External Procedure
  - 동작 과정
  - 생성 절차
  - 사용 예제

제2장 C External Procedure

본 장에서는 C External Procedure의 특징과 동작 과정을 설명한다.

2.1. 특징

C External Procedure는 다음과 같은 특징이 있다.

- 특정 OS에서만 사용할 수 있다.

  Linux, AIX 64bits, HP_IA 64bits, SOLALIS 5.9 64bits, SOLALIS 5.10 64bits에서만 사용 가능하다.

- 임의의 공유 라이브러리를 호출할 수 있다.

  C 프로그래밍 언어에서 동적으로 호출할 수 있는 공유 라이브러리를 작성하는 경우 C External Procedure를 이용하여 호출할 수 있다.

- Tibero 서버와 다른 메모리 영역에서 수행된다.

  사용자가 작성한 공유 라이브러리(이하 **사용자 공유 라이브러리**)는 Tibero 서버의 프로세스(이하 서버 프로세스)가 아닌 **tbEPA**(Tibero C External Procedure Agent) 프로세스에 의해 동적으로 로드된다.

  tbEPA 프로세스의 메모리 영역에서 사용자 공유 라이브러리가 수행되므로, 수행 중에 에러가 발생 하더라도 서버 프로세스에는 영향을 주지 않는다.

- 특권(Privilege)을 부여할 수 있다.

  사용자 공유 라이브러리는 서버 내의 PSM 개체 하나와 대응되므로, 기존에 PSM에 적용할 수 있는 특권을 부여할 수 있다.

- C External Procedure의 실행이 현재 트랜잭션에 포함된다.

  Tibero 서버에서 클라이언트의 요청을 수행하는 중에 C External Procedure의 실행이 요청되면 tbEPA 프로세스는 그 요청을 처리해 줄 때까지 대기한다.

  C External Procedure는 현재 트랜잭션에 포함되어 실행되며, 실행 결과에 따라 현재 트랜잭션을 커밋 또는 롤백시킬 수 있다.

- Callback Service를 이용하여 현재 트랜잭션에 SQL 연산(Operation)을 수행할 수 있다.

  C External Procedure는 현재 트랜잭션에 포함되어 실행되므로, 사용자가 CLI로 작성한 공유 라이브러리 함수를 이용하여 현재 트랜잭션에 질의 또는 DML 등을 수행할 수 있다.

  이러한 기능을 수행하기 위해서는 Tibero에서 제공하는 **Callback Service**를 이용해야 한다. 자세한 내용은 “4.2. Callback Service”를 참고한다.

2.2. 동작 과정

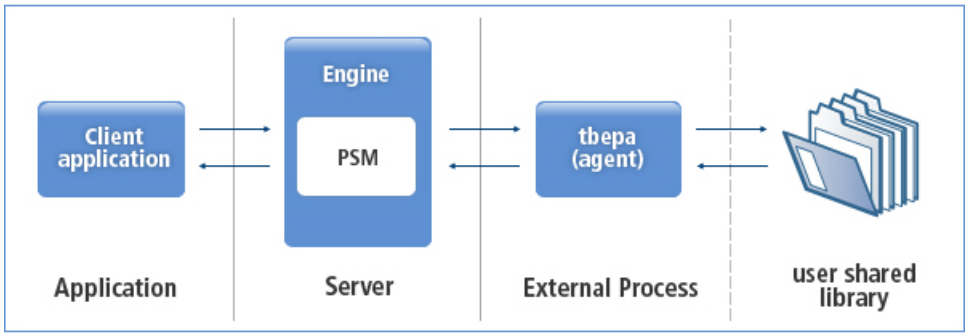
C External Procedure로 등록한 사용자 공유 라이브러리는 수행이 요청되는 순간에 동적으로 로드된다. 이 역할은 서버 프로세스가 아닌 **tbEPA** 프로세스가 담당한다.

tbEPA 프로세스는 C External Procedure의 실행이 요청되면 서버 프로세스로부터 자동으로 생성되며, 서버 프로세스로부터 공유 라이브러리를 호출하기 위한 여러 가지 정보(라이브러리의 위치, 호출할 함수명, 파라미터 정보 등)를 받는다.

그 다음 tbEPA 프로세스는 공유 라이브러리를 동적으로 로드하여 해당 함수를 찾고 사용자가 입력한 파라미터 정보를 통해 함수를 실행시킨다. 만약 반환 값이나 출력 파라미터가 있을 경우 tbEPA 프로세스는 서버 프로세스로 그 값을 전달한다. 이러한 모든 과정은 Tibero에서 자동으로 처리되므로, 사용자는 기존의 PSM 함수를 사용하는 것과 같이 애플리케이션 프로그램을 작성할 수 있다.

다음은 C External Procedure의 동작 과정을 나타내는 그림이다.

[그림 2.1] C External Procedure의 동작 과정



[그림 2.1]를 기준으로 C External Procedure가 동작하는 세부 과정은 다음과 같다.

1. 사용자는 자신이 직접 작성한 사용자 공유 라이브러리(user shared library)를 미리 PSM 개체 하나와 대응시킨다.
2. 사용자가 작성한 애플리케이션 프로그램(Client application)에서 사용자 공유 라이브러리에 대한 실행 요청이 있는 경우 Tibero 서버의 엔진(Engine) 내의 PSM 모듈에서 실행해야 할 해당 라이브러리가 외부에 있음을 확인하고 External Process를 실행시킨다.
3. 2번 과정을 통해 동적으로 로드된 tbEPA 프로세스에 해당 라이브러리의 정보(라이브러리의 위치, 호출할 함수명, 파라미터 정보 등)을 전달하고, 처리 결과를 기다린다.
4. tbEPA 프로세스는 특정 라이브러리를 동적으로 로드하여 함수를 호출하고, 그 결과를 서버로 보낸다.
5. 서버는 tbEPA 프로세스로부터 받은 결과를 분석하여 사용자가 작성한 애플리케이션 프로그램으로 전달한다.
6. tbEPA 프로세스는 해당 세션이 종료될 때 자동으로 종료된다.

제3장 C External Procedure 생성

본 장에서는 C External Procedure를 생성하는 절차를 기본 환경 설정과 생성 절차 단계로 나누어 설명한다.

3.1. 기본 환경 설정

C External Procedure는 이전 장에서도 언급했듯이 tbEPA 프로세스에 의해 수행된다. tbEPA 프로세스를 실행하는 바이너리 파일은 $TB_HOME/client/bin 디렉터리에 존재한다.

본 절에서는 C External Procedure를 생성하기에 앞서 먼저 tbEPA 프로세스를 실행하기 위한 기본 환경을 설정하는 방법을 설명한다.

3.1.1. 초기화 파라미터

tbEPA 프로세스는 처음 실행될 때 사용자가 설정한 초기화 파라미터를 먼저 확인한다. 초기화 파라미터는 $TB_HOME/client/tbepa 디렉터리의 tbepa.cfg 파일에서 설정할 수 있으며, 만약 설정한 초기화 파라미터가 없으면 디폴트로 정의된 속성을 따르게 된다.

tbEPA 프로세스의 실행과 관련하여 사용자가 설정할 수 있는 초기화 파라미터는 다음과 같다.

<<tbepa.cfg>>

```
LOG_DIR=/tmp/epa_log
LOG_LVL=2
MAX_LOG_SIZE=20k
```

초기화 파라미터	설명
LOG_DIR	tbEPA 로그 파일을 저장할 경로이다. (기본값: $TB_HOME/client/tbepa)
LOG_LVL	로그 파일에 남길 로그의 레벨이다. (기본값: 2)
MAX_LOG_SIZE	로그 파일의 최대 크기이다. (기본값: 0, 단위: Byte) – 값이 0인 경우 : 로그 파일의 최대 크기를 설정하는 데 제한이 없다. – 값을 명시한 경우 : 로그 파일이 설정된 최대 크기를 초과하면 로그 파일을 백업한다.
MAX_LOG_BACKUP_SIZE	백업 디렉터리의 로그 파일들의 최대 크기이다. (기본값: 0, 단위: Byte)

초기화 파라미터	설명
	– 값이 0인 경우 : 백업 디렉터리에 있는 로그 파일들의 크기의 합을 설정하는 데 제한이 없다. – 값을 명시한 경우 : 백업 디렉터리에 있는 로그 파일들의 크기의 합이 설정된 최대 크기를 초과하는 경우, 가장 오래된 로그파일 1/3을 삭제한다.

tbepa.cfg 파일에서 초기화 파라미터 LOG_LVL은 1부터 5 사이의 로그 레벨의 값을 설정할 수 있다.

각 로그 레벨에 대한 설명은 다음과 같다.

로그 레벨	로그 레벨명	설명
1	ERROR	에러와 관련된 이벤트이다.
2	WARNING	경고와 관련된 이벤트이다.
3	INFO	함수 호출 등과 관련된 기본적인 트레이스 이벤트이다.
4	DEBUG	디버깅을 위한 구체적인 트레이스 이벤트이다.
5	TRACE	매우 자세한 트레이스 이벤트이다.

3.2. 생성 절차

C External Procedure를 생성하기 위해서는 다음과 같은 세부 절차를 수행해야 한다.

1. 사용자 공유 라이브러리 생성
2. Library Object 등록
3. 사용자 함수를 PSM에 대응
4. 사용자 함수 실행

3.2.1. 사용자 공유 라이브러리 생성

C 프로그래밍 언어에서 동적으로 호출할 수 있는 사용자 공유 라이브러리를 생성한다.

생성하는 순서는 다음과 같다.

1. C 함수를 작성한다.

<<extproc.c>>

```
long find_max(long x, long y)
{
    if (x >= y) return x;
```

```
    else return y;
}
```

2. 컴파일을 수행한다.

   다음은 UNIX 계열(LINUX 포함)에서 컴파일을 수행하는 예이다.

```
cc -g -fpic -shared -o libextproc.so extproc.c
```

3. 컴파일이 완료되면, libextproc.so라는 사용자 공유 라이브러리가 생성된다.

3.2.2. Library Object 등록

Library Object는 사용자 공유 라이브러리 파일을 Tibero에 대응시키는 것으로 일종의 스키마 객체를 의미한다.

Library Object를 등록하는 문법은 다음과 같다.

```
CREATE LIBRARY [schema_name.]library_name
{IS | AS} 'file_path';
```

항목	설명
schema_name	스키마 객체의 이름이다.
library_name	Library Object의 이름이다.
file_path	사용자 공유 라이브러리 파일이 존재하는 절대 경로이다.

참고

CREATE LIBRARY 문을 실행하려면, CREATE LIBRARY라는 권한이 필요하다.

다음은 생성한 사용자 공유 라이브러리 파일을 이용하여 'extproc'이라는 Library Object를 Tibero에 등록하는 예이다.

```
CREATE LIBRARY extproc
IS '/usr/mylib/libextproc.so';
/
```

3.2.3. 사용자 함수를 PSM에 대응

Library Object를 등록한 후 해당 Library Object 내의 특정 함수를 PSM 개체로 등록하면 C External Procedure를 생성할 수 있다. 사용자는 일반적인 PSM를 사용하는 인터페이스 그대로 함수를 호출할 수 있다.

다음은 일반적인 PSM를 선언하는 문법이다.

```
CREATE OR REPLACE {FUCTION | PROCEDURE | PACKAGE}
…
{IS | AS}
LANGUAGE C
LIBRARY library_name
[NAME c_string_literal_name]
[WITH CONTEXT]
[PARAMETERS (external_parameter[, external_parameter]...)];
```

항목	설명
library_name	Library Object의 이름이다.
c_string_literal_name	현재 PSM과 대응할 사용자 함수명이다.
WITH CONTEXT	C External Procedure 유틸리티를 사용하고 있음을 명시할 때 사용한다. 자세한 내용은 “제5장 C External Procedure 유틸리티”, “4.2. Callback Service”를 참고한다.
PARAMETERS (external_parameter)	사용자 함수의 파라미터를 PSM 파라미터와 대응시킬 때 사용한다. external_parameter에 대한 내용은 “4.1. 파라미터 매핑”을 참고한다.

다음은 Library Object 내의 특정 함수(find_max)를 PSM에 대응하는 예이다.

```
CREATE OR REPLACE FUNCTION ext_find_max(num1 BINARY_INTEGER, num2 BINARY_INTEGER)
  RETURN BINARY_INTEGER
  AS LANGUAGE C
  LIBRARY extproc
  NAME "find_max"
  PARAMETERS(num1 int, num2 int);
```

3.2.4. 사용자 함수 실행

사용자 함수는 Tibero상의 하나의 PSM에 대응되므로, PSM의 문법을 사용할 수 있다.

다음은 PSM의 문법을 이용하여 C External Procedure를 실행하는 예이다.

```
SQL> CREATE TABLE TBL (COL1 NUMBER, COL2 NUMBER);
Table 'TBL' created.

SQL> INSERT INTO TBL VALUES(1, 2);
1 row inserted.

SQL> INSERT INTO TBL VALUES(5, 3);
```

```
1 row inserted.

SQL> INSERT INTO TBL VALUES(7, 9);
1 row inserted.

SQL> select col1, col2, ext_find_max(col1, col2) as max from tbl;
      col1       col2        max
---------- ---------- ----------
         1          2          2
         5          3          5
         7          9          9

3 rows selected.
```

제4장 C External Procedure 사용

본 장에서는 C External Procedure에서 PSM 파라미터와 C 파라미터를 매핑하는 방법과 Callback Service를 사용하는 방법을 설명한다.

4.1. 파라미터 매핑

Library Object 내의 특정 함수를 PSM에 대응시키면 사용자 함수의 파라미터로 PSM 파라미터가 그대로 전달된다. 하지만 때로는 PSM으로 전달된 파라미터 중에서 일부만을 사용자 함수로 전달하고 싶거나 그 순서를 변경하고 싶을 때가 있다. 또한 C 프로그래밍 언어의 int, double과 같은 타입의 파라미터로 DBMS에서 사용하는 NULL의 개념을 표현할 수 없을 때도 마찬가지이다.

C External Procedure에서는 이러한 요구를 지원하기 위해 **PARAMETERS** 문법을 제공한다. 즉, 사용자 함수의 파라미터와 PSM 파라미터 간의 파라미터 매핑(Parameter Mapping)을 통해 해결할 수 있다.

본 절에서는 PARAMETERS 문법과 이를 사용하는 방법을 설명한다.

4.1.1. PARAMETERS 문법

PARAMETERS 문법은 생략할 수 있으며, 이 문법을 통해 사용자 함수로 전달할 파라미터와 그 속성을 정의할 수 있다.

PARAMETERS 문법은 다음과 같다.

```
CREATE OR REPLACE {FUCTION | PROCEDURE | PACKAGE}
…
{IS | AS}
LANGUAGE C
LIBRARY library_name
[NAME c_string_literal_name]
[WITH CONTEXT]
[PARAMETERS (external_parameter[, external_parameter]...)];
```

PARAMETERS 문법은 함수의 파라미터 각각을 의미하는 **external_parameter**를 나열하여 구성한다.

또한 external_parameter는 다음과 같이 어떤 방법으로 명시하느냐에 따라 세부 구성요소가 달라진다.

```
{ CONTEXT
  | {parameter_name | RETURN} [property] [BY REFERENCE] [external_datatype]
}
```

- **CONTEXT**

  자세한 내용은 “제5장 C External Procedure 유틸리티”를 참고한다.

- **PSM 파라미터**

  PSM 파라미터는 사용자 함수에 전달할 유형과 형식에 따라 BY REFERENCE, external_datatype 심볼 중에서 설정할 수 있다. PSM 파라미터는 PSM의 데이터 타입이 external_datatype에 대응되는 C 프로그래밍 언어의 데이터 타입으로 순차적으로 변환되어 전달된다.

  다음은 PSM 파라미터를 명시하는 방법을 구성요소별로 설명한다.

  – **parameter_name**

    parameter_name은 사용자 함수로 전달할 PSM 파라미터의 이름을 설정하는 심볼이다.

    다음은 사용자 함수(get_sqrt)로 전달할 PSM 파라미터의 이름(num2)을 설정하는 예이다.

```
CREATE OR REPLACE FUNCTION ext_get_sqrt (num1 BINARY_INTEGER,
                                         num2 BINARY_INTEGER)
       RETURN BINARY_DOUBLE
       AS LANGUAGE C
       LIBRARY extproc
       NAME "get_sqrt"
       PARAMETERS(num2 int);
```

  – **property**

    property는 사용자 함수로 전달할 PSM 파라미터의 특성을 설정하는 심볼이다.

    C의 데이터 타입의 경우 PSM의 데이터 타입과 달리 문자열의 최대 길이나 데이터의 NULL 허용 여부 등을 표시할 수 없다. 따라서 이러한 문제점을 보안하기 위해 C External Procedure에서는 다음과 같은 파라미터의 속성을 지원한다.

속성	C의 데이터 타입	설명
INDICATOR	short	데이터의 NULL 허용 여부를 갖는 지시자(Indicator)이다. – NULL : -1 – NULL이 아닌 경우 : 0
LENGTH	int	문자열 데이터의 현재 길이로 IN, INOUT에서 사용한다.
MAXLEN	int	문자열 데이터의 최대 길이로 OUT, INOUT, RETURN에서 사용한다.

    C의 데이터 타입은 이러한 속성을 파라미터로 전달 받아 사용자 함수를 작성할 때 해당 파라미터의 데이터 타입을 의미한다.

  – **BY REFERENCE**

BY REFERENCE 문은 사용자 함수로 파라미터를 전달할 때 값으로 전달할 것인지 아니면 그 값의 포인터로 전달할 것인지의 여부를 나타내는 심볼이다.

예를 들어 문자열의 경우 C의 데이터 타입으로는 문자열의 값 자체를 담을 수 없기 때문에 char * 형태로 전달할 수 밖에 없다. 또한 출력 파라미터의 경우 그 값이 함수가 수행된 후에 변경되어야 하므로, 그 값에 대한 포인터 타입으로 사용자 공유 라이브러리 함수에 전달되어야 한다.

다음은 간단하게 작성된 C 함수이다.

<<module.c>>

```
void swap(short *x, short *y)
{
    short *z;
    z = x;
    x = y;
    y = z;
}
```

C 함수는 다음의 PSM 함수에서 사용할 수 있다. PSM 함수를 생성하는 문장에서 PARAMETERS 문법의 BY REFERENCE를 명시하면 tbEPA 프로세스는 포인터 형식으로 사용자 함수에 파라미터를 전달한다.

예를 들면 다음과 같다.

```
SQL> CREATE OR REPLACE PROCEDURE ext_swap (x PLS_INTEGER, y PLS_INTEGER)
     IS
     LANGUAGE C
     LIBRARY libextproc
     NAME "swap"
     PARAMETERS(x BY REFERENCE, y BY REFERENCE);
```

- **external_datatype**

external_datatype은 PSM의 데이터 타입을 원하는 형태의 C의 데이터 타입으로 변환하기 위해 사용하는 심볼이다. external_datatype을 생략하면 디폴트로 정의된 C의 데이터 타입으로 변환되어 사용자 함수의 파라미터로 전달된다.

예를 들어 변환이 가능한 C의 데이터 타입으로 전달하는 경우 tbEPA 프로세스 내부에서 사용할 수 있는 타입으로 변환한 후 사용자 함수의 파라미터로 전달한다.

다음은 PSM과 C 함수 간의 변환이 가능한 데이터 타입이다.

PSM의 데이터 타입	C의 데이터 타입(기본)	변환이 가능한 C의 데이터 타입
BINARY_INTEGER PLS_INTEGER	INT	[UNSIGNED] CHAR, SHORT, INT, LONG
FLOAT	FLOAT	FLOAT

PSM의 데이터 타입	C의 데이터 타입(기본)	변환이 가능한 C의 데이터 타입
REAL		
BINARY_DOUBLE	DOUBLE	DOUBLE
CHAR VARCHAR	STRING	STRING, TBSTRING
DOUBLE	TBNUMBER	TBNUMBER
DATE	TBDATE	TBDATE
TIMESTAMP	TBDateTime	TBDateTime
BLOB CLOB	TBLOBLOCATOR	TBLOBLOCATOR

4.1.2. 사용 예제

다음은 PARAMETERS 문법을 사용하여 PSM 함수에 파라미터를 매핑하는 예이다.

1. C 함수를 작성한다.

   <<find_max.c>>

```
short find_max_short(short x, short y)
{
    if (x >= y)
        return x;
    else
        return y;
}
```

2. C 함수의 파라미터를 PARAMETERS 문법을 사용하여 PSM 함수에 파라미터를 매핑한다.

```
SQL> CREATE OR REPLACE FUNCTION ext_find_max_short
      (x PLS_INTEGER, y PLS_INTEGER)     ... ① ...
     RETURN PLS_INTEGER IS
     LANGUAGE C
     LIBRARY libextproc
     NAME "find_max_short"               ... ② ...
     PARAMETERS(x short, y short);
```

① PSM 함수인 ext_find_max_short를 호출한다.

② tbEPA 프로세스는 서버로부터 전달 받은 pls_integer 타입을 short 타입으로 변환하여 사용자 공유 라이브러리 함수인 find_max_short의 파라미터로 보낸다. 그리고 short 타입의 반환 값은 다시 PLS_INTEGER 타입으로 변환하여 서버로 전달한다.

4.2. Callback Service

사용자는 tbCLI 등의 인터페이스를 이용하여 애플리케이션 프로그램을 개발할 수 있다. 또한 DBMS 서버에 접속하여 쿼리, DML 등을 수행할 수 있는 사용자 공유 라이브러리를 만들어 C External Procedure로 사용할 수 있다.

C External Procedure는 서버와 새로운 연결을 맺어 통신하는 일반적인 형태의 tbCLI 프로그램이 아닌 기존의 트랜잭션을 이용하는 Callback Service를 제공한다.

본 절에서는 tbCLI를 이용하여 애플리케이션 프로그램의 개략적인 작성 방법을 살펴본 후 C External Procedure에서 제공하는 Callback Service를 설명한다.

4.2.1. tbCLI를 이용한 애플리케이션 프로그램의 작성 방법

일반적으로 tbCLI를 이용하여 애플리케이션 프로그램을 작성하기 위해서는 다음과 같은 순서로 수행해야 한다.

1. `SQLAllocEnv` 함수를 이용하여 환경 핸들(environment handle)을 생성한다.
2. `SQLAllocConnect` 함수를 이용하여 연결 핸들(connection handle)을 생성한다.
3. `SQLConnect` 함수를 이용하여 DBMS 서버에 접속한다.
4. `SQLAllocStmt` 함수를 이용하여 문장 핸들(statement handle)을 생성한다.
5. 4번에서 할당 받은 문장 핸들에 수행할 작업을 설정한 후 `SQLExecute` 함수를 수행한다.

위 순서에서 1, 2, 3번 과정은 사용자가 작성한 애플리케이션 프로그램을 서버와 연결된 상태로 만들며 4, 5번 과정을 반복하면서 사용자가 원하는 쿼리 또는 DML 등을 수행한다.

참고

자세한 내용은 "Tibero tbCLI 안내서"를 참고한다.

4.2.2. 사용 방법

“4.2.1. tbCLI를 이용한 애플리케이션 프로그램의 작성 방법”과 같은 방식으로 사용자 공유 라이브러리를 작성하면, C External Procedure를 호출할 때 서버와의 새로운 세션을 맺고 사용자 공유 라이브러리를 실행하게 된다. 이때 사용자 공유 라이브러리는 기존에 C External Procedure를 호출한 트랜잭션과는 다른 별도의 트랜잭션으로 동작하게 되며, 또한 기존 트랜잭션의 정보도 볼 수 없게 된다.

C External Procedure에서는 SQLGetExtProcConnect 함수를 이용하여 사용자가 C External Procedure를 호출한 트랜잭션과 같은 트랜잭션 내에서 tbCLI를 이용한 작업을 동시에 수행할 수 있게 해준다.

다음은 SQLGetExtProcConnect 함수를 사용한 예이다.

```
SQLRETURN SQL_API
SQLGetExtProcConnect(ExtProcContext *epCtx, SQLHENV *henv, SQLHDBC *hdbc,
                     SQLSMALLINT *errHdlType, SQLHANDLE *errHdl);
```

*epCtx는 ExtProcContext 구조체에 대한 포인터이며, henv, hdbc는 반환받을 환경 핸들, 연결 핸들을 나타낸다. 그리고 에러 핸들 타입(errHdlType)과 에러 핸들(errHdl)은 에러 발생에 대비하여 정의한 것이다.

SQLGetExtProcConnect 함수를 이용하면 “4.2.1. tbCLI를 이용한 애플리케이션 프로그램의 작성 방법”에서 설명한 순서 중 1번부터 3번까지의 과정을 한 번에 할 수 있고, 그 뒤의 과정은 기존의 tbCLI를 이용한 작업과 동일하게 진행하면 된다.

사용자 공유 라이브러리 함수 내의 DML, 쿼리 등은 현재 사용자 공유 라이브러리 함수를 호출한 트랜잭션 내에서 수행된다.

다음은 그 예이다.

```
SQLRETURN    rc;
SQLHENV      henv;
SQLHDBC      hdbc;
SQLHSTMT     hstmt;
SQLSMALLINT  errHdlType;
SQLHANDLE    errHdl;

char *sql = "insert into tbl values('Tibero')";
SQLINTEGER cnt;

rc = SQLGetExtProcConnect(epc, &henv, &hdbc, &errHdlType, &errHdl);
if (rc != 0) return rc;

rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
if (rc != 0) return rc;

rc = SQLExecDirect(hstmt, sql, SQL_NTS);
if (rc != 0) {
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    return rc;
}

rc = SQLRowCount(hstmt, &cnt);
if (rc != 0)  {
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    return rc;
```

```
}

SQLFreeHandle(SQL_HANDLE_STMT, hstmt);

return rc;
```

참고

위의 예제에서 보듯이 statement handle의 경우 사용자가 직접 SQLAllocHandle()을 통해 만들어야 하고 적절한 곳에서 SQLFreeHandle()을 해야 한다. 하지만 SQLGetExtProcConnect()을 통해 자동으로 생성되는 environment handle과 connection handle은 SQLDisconnect()나 SQLFreeHandle()을 하면 안된다.

4.2.3. 사용 예제

다음은 Callback Service를 이용하여 애플리케이션 프로그램을 작성한 예이다.

```
int connect(ExtProcContext *epc)
{
    SQLRETURN    rc;
    SQLHENV      henv;
    SQLHDBC      hdbc;
    SQLHSTMT     hstmt;
    SQLSMALLINT  errHdlType;
    SQLHANDLE    errHdl;

    char *sql = "insert into psm_test_tab_002 values('TEST')";
    SQLINTEGER cnt;

    rc = SQLGetExtProcConnect(epc, &henv, &hdbc, &errHdlType, &errHdl);
    if (rc != 0) return rc;

    rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if (rc != 0) return rc;

    rc = SQLExecDirect(hstmt, sql, SQL_NTS);
    if (rc != 0) {
        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        return rc;
    }

    rc = SQLRowCount(hstmt, &cnt);
    if (rc != 0) {
        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        return rc;
```

```
    }

    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);

    if (cnt != 1) return 1;

    return 0;
}
```

제5장 C External Procedure 유틸리티

본 장에서는 사용자 공유 라이브러리 함수를 작성하는 데 유용한 C External Procedure 유틸리티를 설명한다.

5.1. 개요

C External Procedure에서는 사용자가 공유 라이브러리 함수를 구현하는 데 도움이 되는 몇 가지 API를 제공하고 있다. 이러한 API를 **C External Procedure 유틸리티**라고 한다.

C External Procedure 유틸리티에 포함된 함수는 다음과 같다.

- `SQLExtProcAllocMemory`
- `SQLExtProcRaiseError`
- `SQLExtProcRaiseErrorWithMsg`

이러한 함수를 사용하기 위해서는 Tibero에서 제공하는 ExtProcContext 구조체를 알아야 하며, 사용자가 공유 라이브러리를 작성할 때와 PSM으로 등록할 때 특별한 처리가 필요하다.

5.2. C External Procedure 유틸리티 사용

C External Procedure 유틸리티를 사용하려면 tbEPA 프로세스에 이 유틸리티를 사용하겠다는 내용을 명시하고, 사용자 공유 라이브러리 또한 이 점을 감안하여 작성해야 한다.

tbEPA 프로세스에 이러한 정보를 전달하기 위해서는 해당 사용자 공유 라이브러리 함수를 PSM으로 등록시킬 때 **WITH CONTEXT** 문법을 명시해야 하며, 사용자 공유 라이브러리 함수의 첫 번째 파라미터는 무조건 **ExtProcContext *로 해야 한다. ExtProcContext는 extproc.h 파일에 정의되어 있으며, tbEPA 프로세스에서 해당 C External Procedure 유틸리티 내의 함수를 처리해주는 채널로 사용한다.

5.2.1. 사용자 공유 라이브러리 함수를 작성할 때 유의 사항

Tibero는 사용자 공유 라이브러리 함수를 작성할 때 C External Procedure 유틸리티를 사용하는 경우를 대비하여 $TB_HOME/client/include 디렉터리에 있는 extproc.h 파일을 제공한다. 만약 사용자가 작성할 공유 라이브러리 함수가 C External Procedure 유틸리티 내의 함수를 사용할 예정이라면 반드시 extproc.h 파일을 포함해야 한다.

예를 들면 다음과 같다.

```
#include "extproc.h"

char * get_encrypted_string(ExtProcContext *epc, char *str)
{
    ...
}
```

5.2.2. 사용자 공유 라이브러리 함수를 PSM으로 등록할 때 유의 사항

"3.2.3. 사용자 함수를 PSM에 대응"을 보면 **[WITH CONTEXT]** 문법을 볼 수 있다. 이것이 바로 C External Procedure 유틸리티를 사용하고 있음을 DBMS에 알려주는 것이고, 이렇게 생성된 PSM을 수행할 경우에는 해당 정보를 tbEPA 프로세스에게 전달한다.

그러면 tbEPA 프로세스는 해당 라이브러리를 로드하여 사용자 함수로 파라미터를 보낼 때 첫 번째 파라미터로 ExtProcContext의 포인터를 전달한다.

```
CREATE OR REPLACE FUNCTION ext_get_enc_char (chr CHAR(10))
       RETURN CHAR
       AS LANGUAGE C
       LIBRARY extproc
       NAME "get_encrypted_string"
       WITH CONTEXT
       PARAMETERS(CONTEXT, chr string);
```

위의 예처럼 PARAMETERS 문법을 명시할 경우 반드시 첫 번째 파라미터는 **CONTEXT**로 해야 한다.

주의

WITH CONTEXT 문법만 명시하고 PARAMETERS 문법을 생략하는 경우 첫 번째 파라미터는 CONTEXT가 되고, 두 번째 파라미터부터 해당 PSM 파라미터가 전달된다는 점에 주의해야 한다.

5.3. C External Procedure 유틸리티

본 절에서는 C External Procedure 유틸리티에서 제공하는 함수를 설명한다.

5.3.1. SQLExtProcAllocMemory 함수

사용자가 작성한 공유 라이브러리 함수에서 동적으로 메모리 할당이 필요한 경우가 있다. 일반적으로 malloc, calloc 등의 함수를 사용하여 런타임 시 메모리를 할당받을 수 있으나, 만약 이 값을 반환 값 등으로 사용할 경우 할당된 메모리를 해제하는 시점이 애매한 경우가 발생한다.

이러한 문제를 해결하기 위해 C External Procedure 유틸리티에서는 SQLExtProcAllocMemory 함수를 제공한다. 이 함수를 사용하면 사용자가 메모리를 동적으로 할당할 수 있으며, 할당된 메모리를 별도로 해제할 필요가 없다. 할당된 메모리는 해당 함수가 실행된 이후에 자동으로 해제된다.

SQLExtProcAllocMemory 함수의 세부 내용은 다음과 같다.

- 문법

```
SQLPOINTER SQLExtProcAllocMemory(ExtProcContext *, size_t size);
```

- 파라미터

파라미터	설명
ExtProcContext *	ExtProcContext 구조체의 포인터이다.
size	할당받을 메모리의 크기이다.

- 반환 값

성공여부	설명
성공	동적으로 할당한 메모리의 시작 주소를 반환한다.
실패	NULL을 반환한다.

- 예제

```
#include <string.h>
#include "extproc.h"
#include "sqlcli.h"

char * my_concat(ExtProcContext *epc, char *str1, char *str2)
{
    char *ret_str;
    uint size;

    size = strlen(str1) + strlen(str2) + 1;
    ret_str = (char *) SQLExtProcAllocMemory(epc, size);

    strcpy(ret_str, str1);
    strcat(ret_str, str2);

    return ret_str;
}
```

5.3.2. SQLExtProcRaiseError 함수

사용자 공유 라이브러리 함수를 수행하는 중에 에러가 발생하거나 에러를 발생시켜야 하는 경우 그 반환 값을 이용하여 PSM 내에서 에러 핸들링 과정을 구현할 수 있다. 단, 사용자가 원하는 로직을 구현하기 위해서는 사용자 공유 라이브러리 함수뿐만 아니라 그 함수와 대응되는 PSM이 사용되는 모든 곳에 에러 핸들링 과정이 필요하다는 단점이 있다.

C External Procedure 유틸리티에서는 사용자 공유 라이브러리 함수에서 직접 DBMS 에러를 발생시켜 앞에서 언급한 단점을 해결할 수 있는 함수를 제공한다. 이 함수를 이용하면 사용자 공유 라이브러리 함수 내에서 DIVIDE BY ZERO, NUMBER EXCEEDS PRECISION 등의 에러를 발생시킬 수 있다. 이러한 기능을 수행하는 함수가 바로 `SQLExtProcRaiseError`이다. 이 함수는 Callback Service에서도 사용할 수 있다.

`SQLExtProcRaiseError` 함수의 세부 내용은 다음과 같다.

- 문법

```
void SQLExtProcRaiseError(ExtProcContext *, int errcode);
```

- 파라미터

파라미터	설명
ExtProcContext *	ExtProcContext 구조체의 포인터이다.
errcode	에러를 발생시킬 에러 코드의 번호이다. 에러 코드에 대한 자세한 내용은 "Tibero 에러 참조 안내서"를 참고한다.

- 예제

```
/* ExtProcContext *epc */
SQLExtProcRaiseError(epc, 5070);
```

5.3.3. SQLExtProcRaiseErrorWithMsg 함수

`SQLExtProcRaiseError` 함수와 같이 에러를 발생시키는 함수이다. 또한 사용자가 정의한 에러도 발생시킬 수 있다.

`SQLExtProcRaiseErrorWithMsg` 함수의 세부 내용은 다음과 같다.

- 문법

```
void SQLExtProcRaiseErrorWithMsg(ExtProcContext *, int errcode, char *errmsg);
```

- 파라미터

파라미터	설명
ExtProcContext *	ExtProcContext 구조체의 포인터이다.
errcode	사용자가 정의한 에러 코드(20000 ~ 20999)이다.
*errmsg	사용자가 정의한 에러 메시지이다.

- 예제

```
/* ExtProcContext *epc */
SQLExtProcRaiseErrorWithMsg(epc, 20100, "User exit procedure");
```

제6장 Java External Procedure

본 장에서는 Java External Procedure의 특징과 동작 과정을 설명한다.

참고

JEPA를 사용하기 위해서는 JDK 1.4 ~ 1.8 버전이 설치되어 있어야 한다.

6.1. 특징

Java External Procedure는 다음과 같은 특징이 있다.

- 임의의 Java 클래스를 호출할 수 있다.

  Java에서 동적으로 호출할 수 있는 클래스를 작성하는 경우 Java External Procedure를 이용하여 호출할 수 있다.

- Tibero 서버와 다른 메모리 영역에서 수행된다.

  사용자가 작성한 클래스(이하 **사용자 Java 클래스**)는 Tibero 서버의 프로세스(이하 서버 프로세스)가 아닌 **JEPA**(Java External Procedure Agent) 프로세스에 의해 동적으로 로드된다.

  JEPA 프로세스의 메모리 영역에서 사용자 Java 클래스가 수행되므로, 수행 중에 에러가 발생 하더라도 서버 프로세스에는 영향을 주지 않는다.

- 권한을 부여할 수 있다.

  사용자 Java 클래스는 서버에 하나의 객체로 등록되므로, DDL 문장과 관련된 권한을 설정할 수 있다. 또한 PSM 함수와도 대응되므로 실행 권한도 설정할 수 있다.

- Java External Procedure는 현재 트랜잭션에 포함된다.

  Tibero 서버에서 클라이언트의 요청을 수행하는 중에 Java External Procedure의 실행이 요청되면 JEPA 프로세스는 그 요청을 처리해 줄 때까지 대기한다.

  Java External Procedure는 현재 트랜잭션에 포함되어 실행되며, 실행 결과에 따라 현재 트랜잭션을 커밋 또는 롤백시킬 수 있다.

- JDBC API를 사용하여 현재 트랜잭션에 SQL 문장을 수행할 수 있다.

  Java External Procedure는 현재 트랜잭션에 포함되어 실행되므로, 사용자가 JDBC로 작성한 함수를 이용하여 현재 트랜잭션에 질의 또는 DML 등을 수행할 수 있다.

  이러한 기능을 수행하기 위해서는 Tibero에서 제공하는 **Internal JDBC Driver**를 이용해야 한다. 자세한 내용은 “8.2. Internal JDBC Driver”를 참고한다.

6.2. 동작 과정

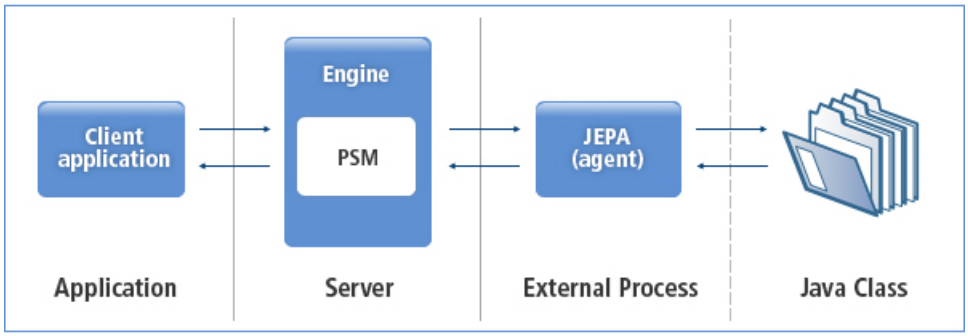
Java External Procedure로 등록한 사용자 Java 클래스는 수행이 요청되는 순간에 동적으로 로드된다. 이 역할은 서버 프로세스가 아닌 **JEPA** 프로세스가 담당한다.

JEPA 프로세스는 설정된 초기화 파라미터에 따라 Tibero 서버가 기동될 때 자동으로 생성되며, 서버 프로세스로부터 사용자 Java 클래스를 호출하기 위한 여러 가지 정보(클래스 이름, 호출할 함수명, 파라미터 정보 등)를 받는다. 그 다음 JEPA 프로세스는 사용자 Java 클래스를 동적으로 로드하여 해당 함수를 찾고 사용자가 입력한 파라미터 정보를 통해 함수를 실행시킨다. 만약 반환 값이나 출력 파라미터가 있을 경우 JEPA 프로세스는 서버 프로세스로 그 값을 전달한다.

이러한 모든 과정은 Tibero에서 자동으로 처리되므로, 사용자는 기존의 PSM 함수를 사용하는 것과 같이 애플리케이션 프로그램을 작성할 수 있다.

다음은 Java External Procedure의 동작 과정을 나타내는 그림이다.

[그림 6.1] Java External Procedure의 동작 과정



[그림 6.1]를 기준으로 Java External Procedure가 동작하는 세부 과정은 다음과 같다.

1. 데이터베이스 서버가 기동될 때 JEPA 프로세스가 생성된다.
2. 사용자는 자신이 직접 작성한 사용자 Java 클래스(Java 소스: .class)를 데이터베이스에 Java 객체로 등록한다.
3. PSM 프로그램을 작성하여 2번에서 등록한 Java 객체와 연결한다.
4. 3번에서 작성한 PSM 프로그램에서 실행 요청이 발생한 경우 JEPA 프로세스와 연결을 맺는다. 그 다음 함수 실행을 위한 정보(클래스 이름, 함수 이름, 파라미터 정보 등)를 전달하고, 처리 결과를 기다린다.
5. JEPA 프로세스는 실행할 함수를 찾아 수행하고, 그 결과를 데이터베이스 서버로 보낸다.
6. 데이터베이스 서버는 JEPA 프로세스로부터 받은 결과를 분석하여 사용자에게 전송한다.
7. 세션이 종료되면 JEPA 프로세스와는 연결이 끊어진다. 그 이후 다음 세션이 연결되고, PSM 프로그램이 호출되면 다시 JEPA 프로세스와 연결된다.
8. JEPA 프로세스는 데이터베이스 서버가 종료될 때 자동으로 종료된다.

제7장 Java External Procedure 생성

본 장에서는 Java External Procedure를 생성하는 절차를 기본 환경설정과 생성 단계로 나누어 설명한다.

7.1. 기본 환경설정

본 절에서는 Java External Procedure를 생성하기에 앞서 기본 환경을 설정하는 방법을 설명한다.

7.1.1. 초기화 파라미터

JEPA 프로세스의 실행과 관련하여 사용자가 설정할 수 있는 초기화 파라미터는 다음과 같다.

<<$TB_SID.tip>>

```
_PSM_BOOT_JEPA=Y
JAVA_CLASS_PATH=/home/tibero/tibero7/instance/tibero/java
```

초기화 파라미터	설명
_PSM_BOOT_JEPA	JEPA 프로세스의 기동 여부를 설정한다. – Y : JEPA 프로세스를 기동하기 위해서는 반드시 이 초기화 파라미터에 '**Y**'를 설정해야 한다. – N : 데이터베이스 서버는 기동이 되지만, JEPA 프로세스는 기동되지 않는다.
JAVA_CLASS_PATH	사용자가 Java 객체를 생성할 때 컴파일 경로로 사용된다. 이 항목이 설정되어 있지 않으면, DB_CREATE_FILE_DEST 초기화 파라미터에 설정된 경로의 하위 디렉터리 Java에 클래스 파일이 생성된다.

7.1.2. JEPA 연결정보

$TB_HOME/client/config 디렉터리에 위치한 tbdsn.tbr 파일에 JEPA 프로세스의 접속 정보를 설정한다.

다음은 tbdsn.tbr 파일에 설정된 접속 정보의 예이다.

<<tbdsn.tbr>>

```
epa=(
      (EXTPROC=(LANG=JAVA)
```

```
            (LISTENER=(HOST=localhost)
                      (PORT=9390)
            )
      )
    )
```

Tibero 서버는 접속 정보를 이용하여 JEPA 프로세스에 연결한다.

7.1.3. JEPA 환경설정

$TB_HOME/client/epa/java/config 디렉터리에 위치한 epa.cfg 파일을 사용자의 시스템 환경에 맞게 수정한다. 환경설정이 적용되면 Java External Procedure를 생성할 준비가 모두 완료된다.

다음은 epa.cfg 파일을 설정한 예이다.

<<epa.cfg>>

```
# listener port
LISTENER_PORT=9390

# initial thread pool size
INIT_POOL_SIZE=10

# max thread pool size
MAX_POOL_SIZE=1000

# gateway encoding "ASCII", "EUC-KR", "MSWIN949", "UTF-8", "UTF-16", "SHIFT-JIS"
ENCODING=UTF-8

#STATIC_LOADING_CLASSES=ex.StaticClass1, ex.st.Class2
```

초기화 파라미터	설명
LISTENER_PORT	tbdsn.tbr 파일에 설정한 포트 번호와 동일하게 설정한다.
INIT_POOL_SIZE	JEPA 프로세스가 기동할 때 처음으로 생성하는 스레드의 개수이다.
MAX_POOL_SIZE	JEPA 프로세스가 생성할 수 있는 최대 스레드의 개수이다.
ENCODING	JEPA 프로세스에서 사용할 인코딩 방식이다. 단, 데이터베이스 서버와 동기화를 위해서 반드시 서로 같은 인코딩을 사용해야 한다.
STATIC_LOADING_CLASSES	JNI를 사용하는 library와 같이 Dynamic Class Loading하기에 적합하지 않은 클래스 라이브러리를 사용할 경우에 필요한 파라미터이다.

참고

tbdsn.tbr 파일과 epa.cfg 파일에 있는 포트 번호는 반드시 일치해야 한다.

7.2. Java External Procedure 생성

Java External Procedure를 생성하기 위해서는 다음과 같은 세부 절차를 수행해야 한다.

1. Java 객체의 생성
2. 생성된 Java 객체를 컴파일
3. PSM 함수의 생성
4. Java 클래스의 실행

7.2.1. Java 객체 생성

Java External Procedure를 생성하기 위해서는 먼저 Java에서 동적으로 호출할 수 있는 형태의 사용자 Java 클래스를 DBMS에 등록해야 한다. 즉, 사용자 Java 클래스를 하나의 데이터베이스 객체로 등록하는 과정이다. 이 과정을 통해 PSM 개체 하나와 사용자 Java 클래스를 대응시키면 기존의 PSM을 호출하는 것과 똑같은 인터페이스로 **사용자 Java 클래스**를 사용할 수 있다.

다음은 Java 객체를 생성하는 예이다.

```
SQL> CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED "JavaExtproc" AS
     public class SimpleMath {
         public static int findMax(int x, int y) {
             if (x >= y) return x;
             else return y;
         }
     }
     /

Java Source 'JavaExtproc' created.

SQL>
```

위와 같은 DDL 문장을 실행하면, 데이터베이스에는 **JavaExtproc**라는 Java 객체가 생성된다. 생성된 Java 객체를 보려면 USER_OBJECTS 뷰나 LS 명령어를 이용하여 조회할 수 있다.

다음은 LS 명령어를 이용하여 생성된 Java 객체를 조회하는 예이다.

```
SQL> LS

NAME                                                      OBJECT_TYPE
--------------------------------------------------------- -------------------
JavaExtproc                                               JAVA

1 row selected.
```

```
SQL>
```

주의

Java External Procedure에서 Java 객체를 호출할 때 빈 생성자(nullary constructor)를 호출하기 때문에 생성자를 생략하거나 다른 생성자(constructor)를 사용할 경우 반드시 빈 생성자(nullary constructor)를 명시해줘야 한다.

7.2.2. 생성된 Java 객체 컴파일

DDL 문장으로 작성된 Java 객체의 소스 코드는 별도의 파일로 생성되며, JAVA_CLASS_PATH 초기화 파라미터에 설정된 경로에 저장된다. Java 객체를 패키지로 생성할 경우 디폴트 저장 경로에 패키지 경로를 생성하여 클래스 파일 형태로 저장한다.

예를 들면 다음과 같다.

```
SQL> CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED "JavaExtproc" AS
     package com.tmax;
     public class SimpleMath {
         public static int findMax(int x, int y) {
             if (x >= y) return x;
             else return y;
        }
     }
```

위 문장을 실행하면 최종적으로 JAVA_CLASS_PATH/com/tmax 디렉터리에 **SimpleMath.class** 파일이 생성된다. 생성된 클래스 파일은 외부 Java 컴파일러를 이용하여 컴파일한다.

외부 Java 컴파일러는 $TB_HOME/bin 디렉터리에 있는 **psmjavac** 스크립트를 사용할 수 있다.

psmjavac 스크립트를 사용하여 실제 컴파일 명령어를 수행하는 문법은 다음과 같다.

```
javac -classpath ${classpath} ${src}
```

항목	설명
classpath	JAVA_CLASS_PATH 초기화 파라미터에 설정된 경로와 **$TB_HOME/client/lib/jar/tibero7-jdbc.jar** 경로가 기본적으로 포함된다.
src	컴파일을 수행할 Java 소스 코드이다.

별도의 외부 라이브러리를 사용하여 Java 객체를 생성할 경우 psmjavac 스크립트 파일을 수정한다.

예를 들어 JVM에서 제공하는 htmlconverter.jar를 사용하는 Java 객체를 생성할 경우에는 다음과 같은 명령어를 수행한다.

```
javac -classpath ${classpath}:$JAVA_HOME/lib ${src}
```

JEPA가 별도의 VM으로 실행되기 때문에 외부 라이브러리를 사용한 객체를 침조하는 PSM 함수를 호출하는 경우 JEPA에서도 해당 라이브러리를 참조하고 있어야 한다. JEPA는 $TB_HOME/client/bin 디렉터리에 있는 tbjavaepa 스크립트에 의해 실행되는데, 다음 exec java 명령 문장의 -classpath 옵션에도 라이브러리의 경로를 추가한다.

```
exec java -verbose:gc -Xms128m -Xmx512m -Djavaepa="$TB_HOME"
      -Dlog4j.configuration=$log4jfile
      -classpath $pool:$collections:$log4j:
       $:$:$config $mainclass CONFIG=$configfile
```

Java 객체를 삭제하기 위해서는 다음과 같은 명령을 사용해야 한다.

```
SQL> DROP JAVA SOURCE "JavaExtproc";
Java Source 'JavaExtproc' dropped.

SQL>
```

7.2.3. PSM 함수 생성

CREATE JAVA 문장을 실행하려면 **CREATE PROCEDURE** 권한이 필요하다.

Java 객체를 다음 예처럼 PSM과 대응시키는 과정을 거치고 나면 PSM 프로그램의 인터페이스를 그대로 사용할 수 있다.

```
SQL> CREATE OR REPLACE FUNCTION find_max(x PLS_INTEGER, y PLS_INTEGER)
     RETURN PLS_INTEGER IS
     LANGUAGE JAVA NAME 'SimpleMath.findMax(int, int) return int';
     /

Function 'FIND_MAX' created.

SQL>
```

PSM 함수를 생성하는 문장에서 PSM과 Java 사이에서 변환하거나 사용할 수 있는 타입은 다음과 같다.

PSM의 타입	Java의 타입
CHAR, LONG, VARCHAR2	– java.lang.String – java.sql.Date – java.sql.Time – java.sql.Timestamp

PSM의 타입	Java의 타입
	– java.lang.Byte – java.lang.Short – java.lang.Integer – java.lang.Long – java.lang.Float – java.lang.Double – java.math.BigDecimal – byte – short – int – long – float – double
DATE	– java.sql.Date – java.sql.Time – java.sql.Timestamp – java.lang.String
NUMBER	– java.lang.Byte – java.lang.Short – java.lang.Integer – java.lang.Long – java.lang.Float – java.lang.Double – java.math.BigDecimal – byte – short – int

PSM의 타입	Java의 타입
	– long – float – double
RAW, LONG RAW	– byte[ ]
ROWID	– java.lang.String

7.2.4. Java 클래스 실행

PSM 함수는 SQL 문장을 실행하는 것과 동일한 방법으로 실행할 수 있다.

다음은 SQL 문장에서 PSM 함수를 실행하는 예이다.

```
SQL> select find_max(4,60) from dual;

FIND_MAX(4,60)
--------------
            60

1 row selected.

SQL>
```

제8장 Java External Procedure 사용

일반적으로 Java를 이용하여 클래스 파일을 생성하고 패키징 과정을 거치면 애플리케이션 프로그램을 개발할 수 있다. 또한 데이터베이스에 단순히 Java 객체를 생성함으로써 별도의 컴파일 및 패키징 과정 없이 다양한 용도의 애플리케이션 프로그램으로 작성할 수도 있다.

본 장에서는 생성된 Java External Procedure를 사용하여 Java 애플리케이션 프로그램을 작성하는 방법을 설명한다.

8.1. 기본적인 Java 애플리케이션 프로그램

8.1.1. 개발 방법

사용자는 일반적으로 Java 애플리케이션 프로그램을 개발하는 방법과 마찬가지로 애플리케이션 프로그램을 작성할 수 있다. 예를 들면 다음과 같다.

```
SQL> CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED "JavaExtproc" AS
     public class SimpleMath {
         public static int findMax(int x, int y) {
             if (x >= y) return x;
             else return y;
         }
     }
     /
```

주의

Java 애플리케이션 프로그램을 개발할 때 주의해야 할 두 가지 사항이 있다.

- 정적(static) 전역 변수를 사용할 수 없으며, 사용할 경우 예상치 못한 오동작이 발생할 수 있다.

- 사용자 Java 클래스를 PSM 함수로 대응시키기 위해서는 반드시 정적 함수로 선언해야 한다.

8.1.2. Java 출력 전송

사용자는 **DBMS_JAVA** 패키지를 사용하여 Java 애플리케이션 프로그램의 출력 내용을 콘솔 화면으로 확인할 수 있다. 예를 들면 다음과 같다.

```
SQL> CREATE OR REPLACE AND RESOLVE Java SOURCE NAMED "JavaExtproc" AS
     public class SimplePrint {
         public static void printMax(int x, int y) {
             if (x >= y) System.out.println("Max="+x);
             else System.out.println("Max="+y);
         }
     }
     /
Java Source 'JavaExtproc' created.

SQL> CREATE OR REPLACE PROCEDURE print_max(x pls_integer, y pls_integer) IS
     LANGUAGE JAVA NAME 'SimplePrint.printMax(int, int)';
     /
Procedure 'PRINT_MAX' created.

SQL> set serveroutput on
SQL> call dbms_java.set_output(2000);

PSM called.

SQL> BEGIN print_max(1, 2); END;
     /
Max=2

PSM completed.

SQL>
```

참고

DBMS_JAVA 패키지에 대한 자세한 내용은 "Tibero tbPSM 참조 안내서"를 참고한다.

8.2. Internal JDBC Driver

사용자는 JDBC 인터페이스를 이용하여 애플리케이션 프로그램을 개발할 수 있는 것처럼 Java 객체 내부에 **Internal JDBC Driver**를 사용할 수 있다. Java External Procedure는 새롭게 데이터베이스 서버와 연결하여 통신하는 일반적인 형태의 JDBC 프로그램이 아닌 기존 트랜잭션을 이용할 수 있는 Internal JDBC Driver를 사용할 수 있다.

8.2.1. 데이터베이스 서버와 연결

일반적인 JDBC 인터페이스의 연결 과정은 다음과 같다.

```
Connection conn = DriverManager.getConnection("jdbc:tibero:localhost:9999:tibero",
                                              "tibero", "tibero");
```

Internal JDBC Driver를 사용하는 방법은 다음과 같다.

```
Connection conn = DriverManager.getConnection("jdbc:default:connection:");
```

기존의 JDBC 인터페이스의 방식을 사용하는 경우에는 현재 트랜잭션에 참여하지 못하고 새로운 세션으로 시작하게 된다. 그러나 Internal JDBC Driver를 사용하는 경우에는 현재 트랜잭션에 참여하여 아직 커밋되지 않은 내용까지도 커밋하거나 롤백할 수 있다.

Internal JDBC Driver를 사용하여 데이터베이스 서버에 연결하고 나면, 일반적인 JDBC 인터페이스를 사용하는 방식 그대로 사용할 수 있다.

8.2.2. 사용 예제

다음은 Internal JDBC Driver를 사용하여 Java 애플리케이션 프로그램을 작성한 예이다.

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED "TestInternal" as
import java.sql.*;
import java.io.*;

public class POManager
{
   public static void addCustomer (int custNo, String custName, String city)
   throws SQLException
   {
       String sql = "INSERT INTO Customers VALUES (?,?,?)";
       try
       {
          Connection conn = DriverManager.getConnection("jdbc:default:connection:");

          PreparedStatement pstmt = conn.prepareStatement(sql);
          pstmt.setInt(1, custNo);
          pstmt.setString(2, custName);
          pstmt.setString(3, city);
          pstmt.executeUpdate();
          pstmt.close();
       }
       catch (SQLException e) {
          System.err.println(e.getMessage());
       }
   }
}
```

색인

Symbols

B

C

E

I

J

L

M

P

S

T

ㅅ

ㅍ