DB Connection Pool

524730-1
2020년 봄학기
6/3/2020
박경신

DB 연동 프로그램에서 문제점

- 데이터베이스 연동 프로그램에서 문제점
  - Transaction
    - 엔터프라이즈 애플리케이션 개발에서 가장 중요한 문제.
  - Connection
    - 애플리케이션과 데이터베이스의 연결을 말함.
    - 사용하지 않는 커넥션으로 인한 데이터베이스 자원낭비.
    - 애플리케이션에서 새로운 DB 연결을 만드는 것은 시스템 부하의 한 요소.
    - 대규모 시스템일수록 커넥션 관리는 중요함

DB Connection Pooling

- DB Connection Pooling
  - 별도의 프로세스로 데이터베이스 커넥션을 관리
  - 일정 수준 이상의 커넥션을 유지하기 때문에 최적의 성능을 보장
  - 사용하지 않는 커넥션의 자동관리 기능
  - 최대 접속에 대한 제한 기능
  - 여러 애플리케이션에 서비스 가능

일반적인 DB Connection Pooling 관리

- 일반적인 DB Connection Pooling 관리 방법
  1. 애플리케이션 서버가 startup 될때 일정수의 커넥션을 미리 생성한다.
  2. 애플리케이션의 요청에 따라 생성된 커넥션 객체를 전달한다.
  3. 일정수 이상의 커넥션이 사용되어 지면 새로운 커넥션을 만든다.
  4. 사용되지 않는 커넥션은 정리하고 최소수의 커넥션을 유지한다.

  - 예전에는 별도로 구현된 커넥션 풀링 소스를 사용.
  - JDBC 2.0 이상에서는 javax.sql.DataSource 인터페이스를 제공.
  - 애플리케이션 서버에서 제공하는 DataSource 구현체를 권장함.

JNDI & DataSource

- JNDI (Java Naming and Directory Interface)
  - 표준화된 디렉토리 및 이름 서비스에 대한 자바 인터페이스.
  - JNDI를 통해 객체참조가 가능함.
  - 애플리케이션에서 DataSource를 JNDI를 통해 제공.

JDBC Connection Pool & DataSource

- JSP에서 DataSource 이용
  - Naming Service를 통해 DataSource 를 구한다.
  - DataSource.getConnection()메서드를 통해 Connection 을 얻는다.
  - 사용한 커넥션은 close()메서드를 이용해 반환한다.

JDBC Connection Pool & DataSource

- Eclipse와 Tomcat9을 연동하여 Mysql DB Connection Pool 설정
  1. Eclipse 의 Serviers/Tomcat v9.0 Server at localhost-config/context.xml 파일 수정

```
<Resource
 name="jdbc/MysqlDB"
 auth="Container"
 type="javax.sql.DataSource"
 driverClassName="com.mysql.cj.jdbc.Driver"
 url="jdbc:mysql://localhost:3306/sampledb?autoReconnect=true"
 username= "xxxx"
 password= "xxxxxxxx"
 testOnBorrow="true"
 maxActive="30"
 maxIdle="3"
 maxWait="3000"/>
```

JDBC Connection Pool & DataSource

Property	Value	Description
JNDI Name	**jdbc/MysqlDB**	Naming 서비스에서 사용할 이름을 입력 일반적으로 DataSource에 jdbc 접두어 사용
Data Source URL	**jdbc:mysql:@localhost:3306/sampled?autoReconnect-true**	DB 접속을 위한 URL 정보를 입력
JDBC Driver Class	**com.mysql.cj.jbdc.Driver**	JDBC 클래스 로딩을 위한 클래스 정보
User Name		DB 사용자 계정
Password		DB 계정의 비밀번호
Max Active Connections	**30**	커넥션 풀에서 관리할 최대 동시 연결 수 지정. Max 값 이상의 연결을 못 만듦
Max Idle Connections	**3**	최대로 유지할 유휴 연결 개수. 설정값 이상의 커넥션은 자동으로 정리됨
Max Wait for Connection	**50000**	Milliseconds (default: 50초) DB 연결을 구하기 위한 대기 시간

JDBC Connection Pool & DataSource

2. 그리고 프로젝트 WEB-INF 아래의 web.xml 파일에 다음 내용 추가

```
<resource-ref>
<description>DB Connection</description>
<res-ref-name>jdbc/MysqlDB</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

3. 그리고 ConnectionPoolProvider의 getConnection() 코드는 아래와 같이 수정

```
import java.sql.Connection;
import java.sql.SQLException;
import javax.naming.Context;
import javax.naming.InitialContext;
//import org.apache.tomcat.jdbc.pool.DataSource;
import javax.sql.DataSource;
```

JDBC Connection Pool & DataSource

```
import java.sql.Connection;
import java.sql.SQLException;
import javax.naming.Context;
import javax.naming.InitialContext;
//import org.apache.tomcat.jdbc.pool.DataSource;
import javax.sql.DataSource;
public class ConnectionPoolProvider {
    public static Connection getConnection() throws SQLException {
    Context context = null; Connection conn = null; DataSource dataSource = null;
    try {
        context = new InitialContext();
        dataSource = (DataSource)context.lookup("java:comp/env/jdbc/MysqlDB");
        conn = dataSource.getConnection();
    } catch(Exception e) {
        System.out.println("getConnection Exception");
        e.printStackTrace();
    }
    return conn;
    }
}
```

Transaction

- Transaction 이란
  - 데이터베이스에서 일련의 작업을 하나로 묶어 처리하는 것.
- 계좌 이체에서의 트랜잭션 문제
  - 한 고객이 a 통장에서 b 통장으로 계좌이체를 하려고 한다.
  - 이때 a 통장에서 돈을 인출하고 b 통장에서 입금하는데 a 통장에서는 정상적으로 처리가 되고 b 통장에서 입금처리에 문제가 발생했다고 하자.
  - 이 경우 a 통장은 어떻게 될까 ?
  - a 통장에서 인출 후 b 통장에 입금이 완료되는 것 까지를 하나의 트랜잭션으로 보고 트랜잭션이 완료되어야 두 작업이 모두 정상적으로 처리 되어야 함.
- 트랜잭션을 개발자가 모두 책임지기에는 너무나 중요한 문제이기 때문에 엔터프라이즈 애플리케이션에서는 미들웨어에서 이 부분을 담당함.
- J2EE 에서는 EJB컨테이너에서 트랜잭션을 관리함.

Transaction Commit & Rollback

- commit
  - 커넥션 단위에서 처리됨.
  - 데이터베이스에서 트랜잭션의 완료를 알리는 명령.
  - commit 이전의 작업 내용은 실제 DB에는 반영이 안됨.
- rollback
  - 커넥션 단위에서 처리됨.
  - 트랜잭션을 취소하는 명령.
  - 현재 트랜잭션 단위에서 수행된 작업을 원 상태로 되돌림.

Transaction Commit & Rollback

- 실습
  - ① 방명록에서 게시물을 등록
  - ② SQL Plus 에서 select 로 등록게시물 확인
  - ③ delete 명령으로 게시물 삭제
  - ④ select 문으로 게시물 삭제 확인
  - ⑤ 웹 브라우저에서 F5 로 다시 읽기 후 게시물 확인

  - 삭제된 게시물이 보임.
  - SQL Plus 에서 commit 명령 수행
  - 다시 브라우저에서 확인
  - 5단계에서 rollback 수행 후 select를 실행
  - 삭제된 게시물이 복구된 것을 확인할 수 있음.
  - SQL Plus 는 자동 commit 모드가 아님.
  - JDBC는 기본이 자동 commit 모드임.

Transaction Commit & Rollback

- auto commit 해제

```
conn.setAutoCommit(false);
```

- JDBC에서 commit 과 rollback
  - connection 을 만든 이후의 모든 작업에 대한 처리.
  - 필요한 처리 부분에서 적절히 활용해야 함.

```
conn.commit();
conn.rollback();
```

Transaction Commit & Rollback

- JDBC에서 Transaction 일괄 갱신
  - JDBC 2.0 추가 기능
  - 여러 갱신 명령을 하나로 묶어 처리하는 방법
  - Statement, PreparedStatement 에서 모두 사용 가능
  - Statement 에서의 일괄 갱신
  - 여러 SQL 문에 대해 하나의 트랜잭션으로 처리.

```
conn.setAutoCommit(false);
Statement stmt = conn.createStatement();
stmt.addBatch("insert into table1 values('1,'test')");
stmt.addBatch("update table2 set memo='test'");
int[] cnt = stmt.executeBatch();
```

Transaction Commit & Rollback

- JDBC에서 Transaction 일괄 갱신
  - PreparedStatement 에서의 일괄 갱신
  - 하나의 SQL문에 대한 여러 데이터 처리

```
PreparedStatement pstmt =
 conn.prepareStatement("update table1 set memo=? where num=?");
pstmt.setString(1,"테스트1");
pstmt.setString(2,"1");
pstmt.addBatch();
pstmt.setString(1,"테스트2");
pstmt.setString(2,"2");
pstmt.addBatch();
int[] cnt = pstmt.executeBatch();
```