JDBC Connection Pool기법을 이용한 DBConnection 성능향상에 관한 연구

최영*, 최황규
강원대학교 컴퓨터정보통신공학부
acad1@lycos.co.kr, hkchoi@kangwon.ac.kr

A Study on Performance Improve of DBConnection Using JDBC Connection Pool Method

Young Choi*, Hwang-Kyu Choi
Dept. of Computer Engineering, Kangwon University

요 약

현재 웹페이지들 대부분은 자료를 데이터베이스에 저장하고 클라이언트측의 특정페이지 요구에 데이터베이스에 연결해 요청에 맞는 리소스를 찾아서 클라이언트측에 다시 보내주는 형식을 취하고 있다. 이러한 일련의 과정 중에서 데이터베이스와 커넥션을 맺는 부분은 매우 많은 시간이 걸리는 작업이다. 본 연구에서는 데이터베이스 접속과 해제를 JDBC Connection Pool 기법을 이용하여 최대한 줄임으로써, 같은 시간에 처리할 수 있는 DBConnection의 성능을 최대한 향상시켰다.

1. 서론

대부분의 웹 어플리케이션은 클라이언트의 특정페이지 요구에 데이터베이스와 연결해 요청에 맞는 리소스를 찾아서 클라이언트에 결과를 보내주고 연결을 끊어 버리는 방식으로 이루어진다. 시간도 많이 걸리고 퍼포먼스도 많이 소요된다. 그러나 DB Connection Pool은 DB Connection Pool 매니저가 일정의 컨넥션을 연결하고 있다가 요청이 들어오면 컨넥션을 할당 해주고 없으면 기다린다. 요청한 클라이언트는 컨넥션을 다쓰면 다시 반납하는 구조로써 속도면이나 퍼포먼스 부분에서 커다란 성능향상을 이룰 수 있다.

예를 들어 일반 프로그램과는 달리 웹을 이용한 수강 신청 작업과 같은 경우 검색 작업이 많아, 많은 학생들이 동시에 DB를 access하므로 과부하현상이 발생한다. 이로 인해 시스템의 성능 저하와 다운현상, 서버의 접속 불가, 자료 처리의 지연 등 예기치 않은 많은 문제점이 발생할 수 있다.

한 애플리케이션이 데이터베이스를 이용하여 작업을 수행할 때 가장 데이터베이스에 부하를 많이 주는 작업은 데이터베이스 접속과 해제인데, 이 접속과 해제를 최대한 줄이는 방법만으로도 데이터베이스의 전체적인 성능 향상을 쉽게 꾀할 수 있다.

이에 본 연구에서는 JAVA기반의 JDBC Connection Pool기법을 이용하여 데이터베이스와 연결된 객체들을 미리 준비했다가 사용자가 필요할 때 준비된 객체를 사용하도록 보내주고 사용이 끝나면 연결객체를 다시 반납 받는 방법을 이용하여 DB access로 인한 서버의 과부하와 자료처리 지연을 방지하고, 시스템 자원을 효율적으로 운영함으로써 시스템 전체 성능향상을 이루고자 하였다.

2. 데이터베이스연결 관리

2.1 Improved Performance with a Connection Pool

본 논문에서 소개할 Connection pool은 여러개의 데이타베이스 connection을 유지하고 manager 클래

스는 여러개의 connection pool을 사용할 수 있도록 interface를 제공해준다.

DBConnectionPool 클래스는 다음과 같은 메쏘드를 제공한다.

- Pool에서 데이타베이스와 연결된 커넥션 얻기
- Pool에 커넥션 반환
- Shutdown시에 모든 리소스와 커넥션을 릴리스

time-out, communication failure, 그 외 여러가지 원인으로 인한 연결실패를 관리하고, 미리정의된 값으로 최대 연결갯수를 제한 할 수도 있다.

한편 여러개의 커넥션풀을 관리하는 DBConnection Manager 클래스는 다음과 같은 기능을 제공한다.

- JDBC driver를 로딩하고 등록
- properties 파일에 정의된 대로 DBConnectionPool 객체를 생성
- 커넥션풀 네임을 DBConnection 인스턴스에 매핑
- 모든 풀을 안전하게 셧다운 할수 있도록, 커넥션 풀을 사용하는 클라이언트를 모니터링 한다.[1][2][3]

2.2 The DBConnectionPool class

DBConnectionPool 클래스는 하나의 데이타베이스에 연결된 풀을 대변한다. 데이타베이스는 protocol identifier, driver identifier, database identifier로 구성된 JDBC URL로 표현된다. 이것에 더해서 풀은 클라이언트에서 사용될 이름과 사용자이름, 패스워드, 최대 연결갯수 정보를 가지게 된다.

DBConnectionPool의 생성자는 위에서 언급한 모든 값을 파라메타로 받는다.

```
public DBConnectionPool(String url,String id,String
        pass){    this.url=url;
                  this.id=id;
                  this.pass=pass; }
```

이것은 모든 파라메타값을 내부변수로 저장한다. DBConnectionPool 클래스는 여기서 커넥션을 얻기 위한 두가지 방식을 제공한다. 두 경우 모두 현재 가용한 커넥션이 있으면 그것을, 없으면 새로운 커넥션을 생성해서 리턴 해준다. 만약 현재 가용한 것이 없고 최대 연결 값에도 도달했다면 첫번째 메쏘드는 null을, 두번째 메쏘드는 가용한 연결이 생길때까지 기다린다.

```
public synchronized Connection getConnection() throws
    Exception { Connection connection=null;
                while(connStack.size()<=0){
                 if(loanConnNum<maxConnNum)
    {   connection=makeConnection();    break;
                } else
                    wait();
                    } if(connection==null){
    connection=(Connection)connStack.get(0);
                    connStack.removeElementAt(0);
 if(connection.isClosed()) connection=makeConnection();
            }   loanConnNum++;
                return connection; }
```

풀에 있는 가용한 모든 커넥션 객체는 connStack 이라는 Vector에 유지된다. 이 벡터에 하나라도 커넥션 객체가 있으면 makeConnection()은 첫 번째 연결을 끄집어낸다. 커넥션 객체는 벡터의 마지막에 새로 붙기 때문에 이처럼 앞에 것을 끄집어 내는 것은 inactivity에 의한 DB와의 disconnect로 인해 발생될 위험을 최소화한다.

클라이언트에게 커넥션 객체를 리턴해주기 전에 isClosed() 메쏘드를 호출해 커넥션 객체가 가용한 것 인지 아닌지를 검사한다. 만약 커넥션이 닫혔다거나 exception이 발생하면 다른 커넥션 객체를 얻기 위해 메쏘드를 재호출한다.

connStack 벡터에 가용한 연결이 없으면 최대값에 도달했는지 체크한다. maxConnNum 값이 0이면 "no limit"를 의미한다. 최대값 제한이 없거나 최대 값에 아직 도달하지 않은 상태면 새로운 커넥션을 생성한다. 생성에 성공하면 loanConnNum의 커넥션의 수(클라이언트에게 넘겨준 갯수)를 하나 늘리고 클라이언트에게 커넥션 객체를 리턴 해준다. 실패하면 null 을 리턴한다. makeConnection() 메쏘드는 새로운 커넥션객체를 생성한다. private 메쏘드이고 id와 password 값이 있던 없던 커넥션을 생성한다.

```
 private Connection makeConnection() throws
   Exception{    Connection connection=
       DriverManager.getConnection(url,id,pass);
                 return connection; }
```

이것은 JDBC DriverManager로부터 커넥션객체를 얻는다. DBConnectionPool에서 커넥션객체를 받아쓴 것은 커넥션객체를 풀에 되돌려주어야 한다. returnConnection() 메쏘드는 되돌려주는 커넥션객체를 파라메타로 받는다.

```
     public synchronized void
returnConnection(Connection connection){
            connStack.add(connection);
            loanConnNum--;    notifyAll();   }
```

커넥션객체는 connStack 벡터 마지막에 더해지고 loanConnNum 커넥션 갯수를 하나 줄인다.

notifyAll()을 호출함으로써 가용한 커넥션객체가 생기기를 기다리고 있을지 모를 클라이언트에게 알려준다.

대부분의 서블릿엔진은 단계적인 셧다운을 위한 방법을 제공한다. 데이타베이스 커넥션 풀은 모든 커넥션을 올바르게 닫기 위해 이러한 이벤트에 대해 notified 되어야한다.

셧다운시의 관리책임은 DBConnectionManager에게 있지만 실질적으로 풀의 모든 커넥션을 닫는 작업은 DBConnectionPool이 담당한다.

DBConnectionPool의 close()메쏘드는 DBConnection Manager에 의해 호출된다.

```
public synchronized void close() throws Exception{
        for(int i=0;i<connStack.size();i++){
                Connection connection=
                (Connection)connStack.get(i);
                connection.close();
        }    connStack.removeAllElements();} }
```

이것은 루프를 돌면 connStack 벡터의 모든 커넥션을 닫는다. 커넥션이 모두 닫히고 나면 벡터에서 모든 커넥션을 제거한다.[1][2][3]

2.3 The DBConnectionManager class

DBConnectionManager 클래스는 Singleton 패턴을 적용해 구현되었다. Singleton이란 오직 한개의 인스턴스만이 존재함을 의미한다. 다른 객체들은 static 메쏘드(class method)를 통해 이것의 인스턴스를 가르키는 레퍼런스를 얻게된다.

DBConnectionManager를 이용하는 클라이언트측은 getInstance() 메쏘드를 통해 싱글 인스턴스의 레퍼런스를 얻는다.

```
public static synchronized DBConnectionPoolMgr
        getInstance() throws Exception{
                if(instance==null)
    nstance= new DBConnectionPoolMgr();
                return instance;    }
```

싱글 인스턴스는 이 메쏘드가 최초 호출될때 생성되고 static 변수 instance에 저장된다.

DBConnectionPoolMgr() 메쏘드는 JDBC driver를 로딩하고 등록한다.

```
private DBConnectionPoolMgr() throws Exception{
Class.forName
("com.microsoft.jdbc.sqlserver.SQLServerDriver");
String url="jdbc:microsoft:sqlserver://localhost:1433";
        String id="sa";
        String pass="";
        pool=new DBConnectionPool(url,id,pass);
    }
```

DBConnectionManager는 getConnection()과 returnConnection() 메쏘드를 클라이언트측에 제공한다. 이것들은 풀 이름을 파라메타로 받으며, 같은 풀 객체를 릴레이 방식(얻고->되돌리고->얻고->되돌리고)으로 호출한다.

```
public Connection getConnection() throws Exception{
                Connection connection=
                 pool.getConnection();
                return connection;}
public void returnConnection(Connection connection)
 throws Exception{ pool.returnConnection(connection);
}
```

마지막으로, DBConnectionManager에는 close()라는 것이 있다. 이것은 shutdown을 위해 사용된다. 이미 보았다시피 DBConnectionManager를 사용하는 각각의 클라이언트는 getInstance()를 통해 레퍼런스를 얻고 이때 DBConnectionManager는 클라이언트를 카운트한다. 각각의 클라이언트가 셧다운시 Close()를 호출하고 이때는 클라이언트 카운트가 감소하게 된다. 마지막 클라이언트에서 Close()를 호출하면 DBConnectionManager는 모든 데이타베이스 연결을 끊기 위해 DBConnectionPool 객체의 Close()를 호출한다.[1][2][3]

```
public synchronized void close() throws Exception{
                pool.close();        }
```

3. 수강 신청 관리 시스템 구현

프로그램의 개발은 데스크 탑 PC를 사용하였으며, 시스템 구현 환경은 표와 같다.

표 1. 웹 어플리케이션 개발운영환경

장치	개발운영환경
운영체제	Windows 2000 Server
응용 프로그램	MS-Sql Server2000 j2sdk1.4.0_01 Apache-tomcat4 IIS5.0
개발 어플리케이션 URL	http://sugang.dongu.net

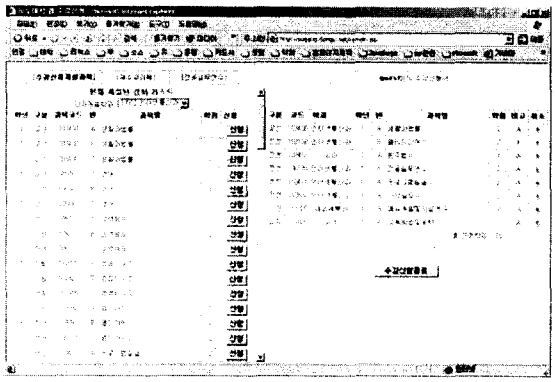

그림1. 수강신청화면

4. 성능평가

- 매번 직접 JDBC Driver 연결하는 경우

java HttpConn http://localhost/db_jdbc.jsp -c <동시유저수> -n <호출횟수> -s 0

TOTAL(1,10) iteration=10 ,
average=249.40 (ms), TPS=3.93
TOTAL(50,10) iteration=500 ,
average=9,051.84 (ms), TPS=4.83
TOTAL(50,10) iteration=500 ,
average=9,149.84 (ms), TPS=5.01
TOTAL(100,10) iteration=1000 ,
average=17,550.76 (ms), TPS=5.27
TOTAL(200,10) iteration=2000 ,
average=38,479.03 (ms), TPS=4.89
TOTAL(300,10) iteration=3000 ,
average=56,601.89 (ms), TPS=5.01

- DB Connection Pooing 을 사용하는 경우

java HttpConn http://localhost/db_pool_cache.jsp -c <동시유저수> -n <호출횟수> -s 0

TOTAL(1,10) iteration=10 ,
average=39.00 (ms), TPS=23.26
TOTAL(1,10) iteration=10 ,
average=37.10 (ms), TPS=24.33
TOTAL(50,10) iteration=500 ,
average=767.36 (ms), TPS=45.27
TOTAL(50,10) iteration=500 ,
average=568.76 (ms), TPS=61.26
TOTAL(50,10) iteration=500 ,
average=586.51 (ms), TPS=59.79
TOTAL(50,10) iteration=500 ,
average=463.78 (ms), TPS=67.02
TOTAL(100,10) iteration=1000 ,
average=1,250.07 (ms), TPS=57.32
TOTAL(200,10) iteration=1462 ,
average=1,875.68 (ms), TPS=61.99
TOTAL(300,10) iteration=1824 ,
average=2,345.42 (ms), TPS=61.51
NOTE: average:평균수행시간, TPS:초당 처리건수

즉, JDBC Driver 를 이용하여 직접 DB연결을 하는 구조는 1초에 5개의 요청을 처리할 수 있는 능력이 있는 반면, DB Connection Pooling 을 사용할 경우는 초당 60여개 요청을 처리할 수 있는 것으로 나타났다.

5. 결론 및 향후 전망

지금까지 데이터베이스 연결을 효율적으로 관리할 수 있는 JDBC Connection Pool을 이용한 웹 어플리케이션을 구축하였다.

데이터베이스 관리는 웹 개발과 유지에 있어서 매우 중요한 부분이다. 아울러 데이터베이스 연결은 시스템 전체의 성능을 좌우하는 매우 중요한 요소라고 할 수 있다.

본 논문에서 구현한 웹 어플리케이션은 빈번한 데이터베이스 연결과 해제로 인한 성능저하와 DB access로 인한 서버의 과부하, 자료처리 지연을 방지하고, 시스템 성능을 향상시킬 수 있었다.

현재 데이터베이스 연결관리를 위한 다양한 JDBC Connection Pool을 만들거나, TCP/IP Socket Pool을 만드는 등 Object Pool Control 기법 들이 개발되고 있다.

참고문헌

[1] Sun Microsystems, "Designing Enterprise Applications with the Java™2 Platform, Enterprise Edition",2000

[2] Sun Microsystems, "The JavaTM2 Enterprise Edition Developer's Guide",2000

[3] 권혜윤 외 1, JSP 영진닷컴 2002

[4] Forte for Java, Ver. 3.0, Enterprise Edition, Sun Microsystems, 2001

[5] Richard Monson-Haefel,"Enterprise Java Beans",3rd Ed.,O'Reilly & Associates, 2001