솔루션개발지원
[제목: (주)대길소프트 JDBC Driver 개발지원]

한국소프트웨어진흥원
공개SW기술지원센터

<Revision 정보>

일자	VERSION	변경내역	작성자
2007.07.09	0.1	초기 작성	허종윤

목 차

<그림 차례>

1. 문서 개요

본 문서는 공개SW 기술지원 센터에서의 일반 기업 및 기관 대상의 솔루션개발지원 결과를 보고하기 위해 제작되었다.

가. 문서의 목적

다음과 같은 세부적인 목적을 달성하기 위하여 작성되었다.

○ 개발지원 요청내용 파악

○ 개발지원 수행을 통하여 솔루션의 공개SW 기반 전환을 위한 개발프로젝트 진행시 기술적 측면과 프로젝트 수행측면의 참조자료로 활용

나. 본 문서의 사용방법

다음과 같은 방법으로 사용할 수 있다.

○ 본 시스템의 기능 구조도를 참조하여 공개SW기반 JDBC Driver 개발시 필요한 작업량 예측, 사전요구 기술요소, 시스템 아키텍쳐 구성을 참조한다.

○ 공개SW기반 시스템으로 전환하는 각 요소들의 구성내역과 개발지원내용을 참조한다.

다. 참고사항

○ 개발지원시 시스템 관련사항, 기술적 배경 등 참고사항들을 기술한다.

2. 개발지원 요청내역

가. 대상 기관

구 분		비 고
기관명	(주)대길소프트	
담당자	오광일팀장	
연락처	02-997-1698	
회사 위치	서울시 도봉구 쌍문2동 576	

나. 요청 사항

항목	요청 내용	지원 내역	비 고
	1. 공개SW기반 JDBC Driver 개발 2. DB Connection Pool 기능개선	1. 공개SW기반 개발 과정에서 요구되는 개발가이드 제공 2. 공개SW기반 전환을 위한 시스템 아키텍쳐 컨설팅 및 구성 3. JDBC Driver 개발 지원	

다. 지원인력 구성

담당	직급	성명	소속사	인력 구분	기간	지원내용	비 고
개발 지원	차장	허종윤	OSTSC	상주	2007.07.09 ~ 2007.07.27	- 시스템 아키텍쳐 구성 - 공개SW 개발 컨설팅 및 가이드 제공	

3. 개발지원 내용

가. 개발가이드 제공

한국소프트웨어진흥원에서 제공하는 “실전웹표준가이드”를 기초로 하여 공개SW 기반 APP로 개발 과정에서 요구되는 개발가이드를 제시하였으며 주요 내용은 아래와 같다.

- XHTML 가이드 제공
- CSS 레이아웃 가이드 제공
- DOM/Script 가이드 제공
- 표준 웹 프로그래밍 가이드 제공

나. 개발지원 범위

개발지원 범위는 Oracle의 Reference Cusor 기능개선을 위하여 JDBC용 JAVA PACKAGE 파일중 elsjdbc.class 개발 지원과 DB Connection Pool 기능개선에 있다. 전체 JAVA PACKAGE는 <그림 1. JAVA PACKAGE>과 같다.

- elsoft
  - eftpmanager
    - EftpFileMng.class
  - extquery
    - extquery.class
    - RoadRowData.class
  - jdbc
    - ArrayParam.class
    - DBMSTypes.class
    - elsjdbc.class
    - EncodingTypes.class
    - escapeDateTime.class
    - getBindObj.class
    - HanTypes.class
    - jBindBase.class
    - jBindBlob.class
    - jBindBool.class
    - jBindByte.class
    - jBindClob.class
    - jBindDate.class
    - jBindDouble.class
    - jBindFloat.class
    - jBindLong.class

- jBindLongBinary.class
- jBindShort.class
- jBindText.class
- jBindTime.class
- jBindTimeStamp.class
- MakeSQLStatement.class
- MultiQueryData.class
- OdbcTypes.class
- PoolConnect.class
- QueryDef.class
- ReadData.class
- RunMultiSQL.class
- RunMultiSQLMng.class
- SQLProceCol.class
- SQLProcedure.class

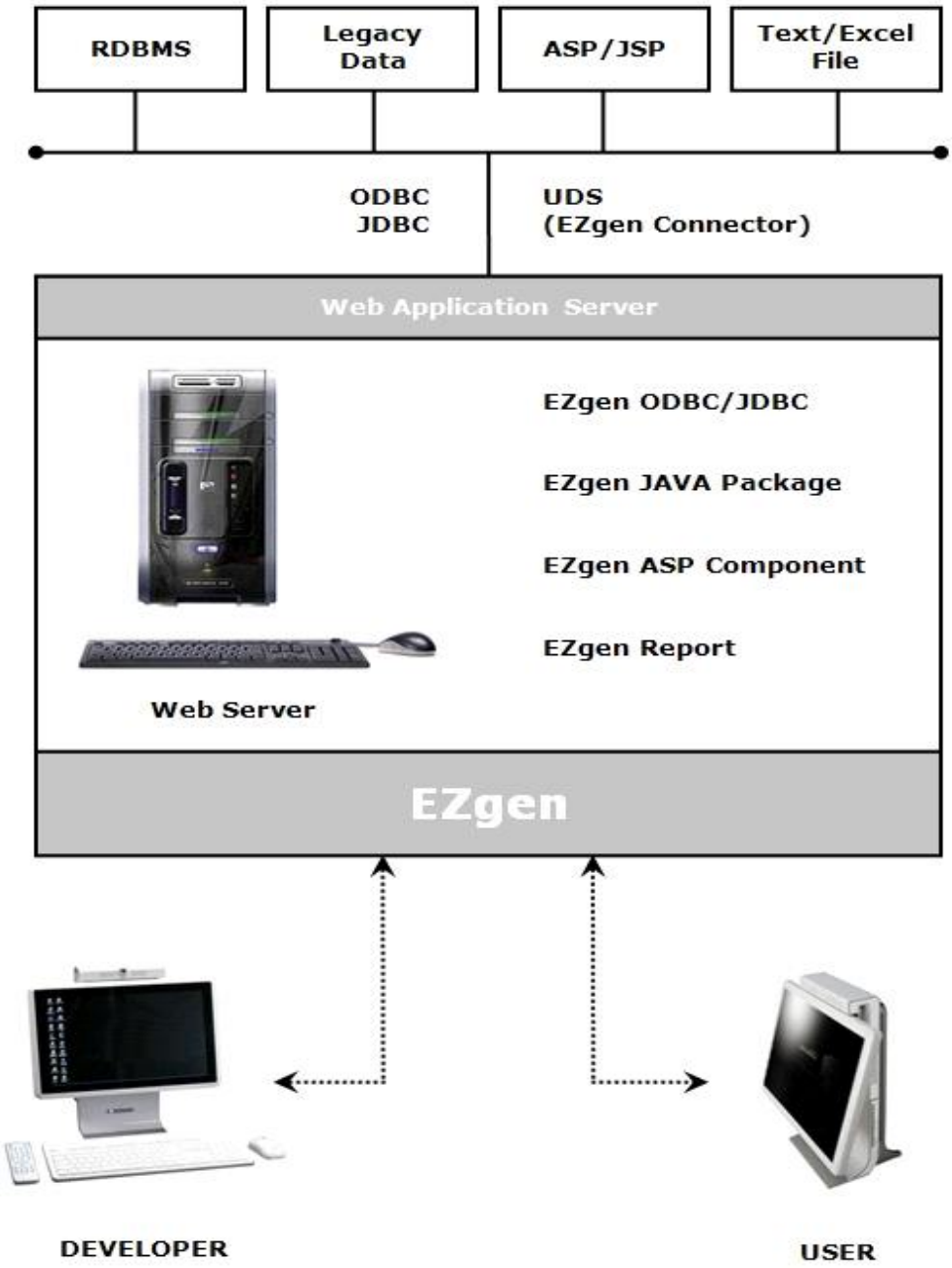
- serverinfo
  - serverinfo.class
- uploadfile
  - HttpUploadFileMng.class
  - PathAlias.class
  - SessionFile.class
- unicode
  - eftpmanager
    - EftpFileMng.class
  - extquery
    - extquery.class
    - RoadRowData.class
  - jdbc
    - ArrayParam.class
    - DBMSTypes.class
    - elsjdbc.class
    - EncodingTypes.class
    - escapeDateTime.class
    - getBindObj.class
    - HanTypes.class
    - jBindBase.class
    - jBindBlob.class
    - jBindBool.class
    - jBindByte.class

- jBindClob.class
- jBindDate.class
- jBindDouble.class
- jBindFloat.class
- jBindLong.class
- jBindLongBinary.class
- jBindShort.class
- jBindText.class
- jBindTime.class
- jBindTimeStamp.class
- MakeSQLStatement.class
- MultiQueryData.class
- OdbcTypes.class
- PoolConnect.class
- QueryDef.class
- ReadData.class
- RunMultiSQL.class
- RunMultiSQLMng.class
- SQLProceCol.class
- SQLProcedure.class

- serverinfo
  - serverinfo.class
- uploadfile
  - HttpUploadFileMng.class
  - PathAlias.class
  - SessionFile.class

<그림 1. JAVA PACKAGE>

4. 개발내역



<그림 2. 시스템 구조도>

시스템 구조도는 <그림 2. 시스템 구조도>와 같이 JAVA PACKAGE 파일이 JDBC 드라이버를 이용하여 데이터베이스와 연동하는 구조로 되어있다. 본 문서에서는 JAVA PACKAGE 파일중 elsjdbc.java 개발지원과 DB Connection Pool 기능개선에 있다.

가. JDBC Driver 개발지원

Oracle의 Reference Cusor 기능개선을 위하여 다음과 같이 eljjdbc.java 파일을 수정하였다.

```
package elsoft.jdbc;

import java.io.*;
import java.sql.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import oracle.jdbc.driver.OracleCallableStatement;

// Referenced classes of package elsoft.jdbc:
//             PoolConnect, QueyrDef, getBindObj, jBindBase,
//             MakeSQLStatement, SQLProceCol, SQLProcedure, RunMultiSQLMng,
//             MultiQueryData, RunMultiSQL, ArrayParam

public class elsjdbc
    implements Runnable
{

    public elsjdbc()
    {
        m_stDriverClass = null;
        m_stUID = null;
        m_stPWD = null;
        m_nMaxThreads = 3;
        m_nTimeOut = 60;
        m_nInEncoding = 0;
        m_nOutEncoding = 0;
        m_stEncoding = "KSC5601";
        m_bTrim = false;
        m_nCheckedOut = 0;
        m_Connections = new Vector();
        m_SessionConnPool = new Hashtable();
        m_DNSFilters = new Vector();
```

```
        m_bDNSFiltere = false;
        m_bReplaceIgnoreCase = true;
        m_stReplaceOld = null;
        m_stReplaceNew = null;
        m_nLogType = 0;
        m_stLogFile = null;
        m_Logger = null;
        m_nMonitorInterval = 0x927c0L;
        m_nPoolTimeout = 0x2dc6c0L;
        m_nCheckPool = 0;
        m_ConnLogger = null;
    }

    public elsjdbc(int i, int j)
    {
        m_stDriverClass = null;
        m_stUID = null;
        m_stPWD = null;
        m_nMaxThreads = 3;
        m_nTimeOut = 60;
        m_nInEncoding = 0;
        m_nOutEncoding = 0;
        m_stEncoding = "KSC5601";
        m_bTrim = false;
        m_nCheckedOut = 0;
        m_Connections = new Vector();
        m_SessionConnPool = new Hashtable();
        m_DNSFilters = new Vector();
        m_bDNSFiltere = false;
        m_bReplaceIgnoreCase = true;
        m_stReplaceOld = null;
        m_stReplaceNew = null;
        m_nLogType = 0;
        m_stLogFile = null;
        m_Logger = null;
        m_nMonitorInterval = 0x927c0L;
        m_nPoolTimeout = 0x2dc6c0L;
        m_nCheckPool = 0;
        m_ConnLogger = null;
```

```
        m_nMonitorInterval = i * 1000;
        m_nPoolTimeout = j * 1000;
        Thread thread = new Thread(this);
        thread.setDaemon(true);
        thread.start();
    }

    public void run()
    {
        do
        {
            try
            {
                Thread.currentThread();
                Thread.sleep(m_nMonitorInterval);
            }
            catch(InterruptedException interruptedexception) { }
            RunThread();
        } while(true);
    }

    public synchronized void RunThread()
    {
        Calendar calendar = m_ConnLogger != null ? Calendar.getInstance() : null;
        Vector vector = new Vector();
        long l = System.currentTimeMillis();
        Enumeration enumeration = m_Connections.elements();
        do
        {
            if(!enumeration.hasMoreElements())
                break;
            PoolConnect poolconnect = (PoolConnect)enumeration.nextElement();
            if(l - poolconnect.m_lasttouch >= m_nPoolTimeout)
            {
                vector.addElement(poolconnect);
                try
                {
                    if(poolconnect.m_conn != null && !poolconnect.m_conn.isClosed())
                        poolconnect.m_conn.close();
```

```
                }
                catch(SQLException sqlexception) { }
            }
        } while(true);
        int i = vector.size();
        if(i > 0)
        {
            PoolConnect poolconnect1;
            for(Enumeration enumeration1 = vector.elements();
enumeration1.hasMoreElements(); m_Connections.removeElement(poolconnect1))
                poolconnect1 = (PoolConnect)enumeration1.nextElement();

            vector.removeAllElements();
        }
        if(m_ConnLogger != null)
        {
            String s = new String("PoolTimeOutCnt:");
            s = s + i;
            WriteLogPool(calendar, s);
        }
        notifyAll();
    }

    public void init(String s, int i, int j, String s1, String s2, String s3)
    {
        m_stDriverClass = s;
        m_nMaxThreads = i;
        m_nTimeOut = j;
        m_stURL = s1;
        m_stUID = s2;
        m_stPWD = s3;
    }

    public void UseEncoding(int i, int j, String s, boolean flag)
    {
        m_nInEncoding = i;
        m_nOutEncoding = j;
        m_stEncoding = s;
        m_bTrim = flag;
```

```
    }

    public void SQLReplace(String s, String s1, boolean flag)
    {
        m_stReplaceOld = s;
        m_stReplaceNew = s1;
        m_bReplaceIgnoreCase = flag;
    }

    public void setLogFile(int i, String s)
    {
        m_nLogType = i;
        if(m_nLogType == 0)
        {
            m_Logger = null;
            return;
        }
        if(m_nLogType == 1 || s == null)
        {
            m_nLogType = 1;
            m_Logger = new PrintStream(System.out);
        } else
        {
            try
            {
                m_stLogFile = s;
                FileOutputStream fileoutputstream = new
FileOutputStream(m_stLogFile);
                m_Logger = new PrintStream(fileoutputstream, true);
            }
            catch(Exception exception)
            {
                System.err.println("Unable to open EZgen log file named ₩"" +
m_stLogFile + "₩" using System.out instead");
                m_nLogType = 1;
                m_Logger = new PrintStream(System.out);
            }
        }
    }
```

```
    public void setLogConnect(int i, String s)
    {
        if(i == 0)
        {
            m_ConnLogger = null;
            return;
        }
        if(i == 1 || s == null)
            m_ConnLogger = new PrintStream(System.out);
        else
            try
            {
                FileOutputStream fileoutputstream = new FileOutputStream(s);
                m_ConnLogger = new PrintStream(fileoutputstream, true);
            }
            catch(Exception exception)
            {
                System.err.println("Unable to open EZgen log file named ₩"" + s +
"₩" using System.out instead");
                m_ConnLogger = new PrintStream(System.out);
            }
    }

    public void addDNSFilter(String s)
    {
        m_DNSFilters.addElement(s);
        m_bDNSFiltere = true;
    }

    public void RunQuery(HttpServletRequest httpservletrequest, HttpServletResponse
httpservletresponse)
        throws ServletException, IOException
    {
        QueyrDef queyrdef = new QueyrDef();
        queyrdef.m_nExecCount = 0;
        ServletOutputStream servletoutputstream =
httpservletresponse.getOutputStream();
        if(m_stDriverClass == null)
```

```
            {
                queyrdef.WriteError("Can't Define DriverClass Name",
servletoutputstream);
                return;
            }
            if(!GetInPutString(httpservletrequest, servletoutputstream, queyrdef))
                return;
            PoolConnect poolconnect = null;
            boolean flag = false;
            String s = null;
            if(queyrdef.m_nQSession >= 0)
            {
                flag = true;
                HttpSession httpsession = httpservletrequest.getSession(true);
                if(httpsession != null)
                    s = httpsession.getId();
                if(httpsession == null || s == null)
                {
                    queyrdef.WriteError("Can't getSession()", servletoutputstream);
                    return;
                }
                poolconnect = getSessionPoolConnect(s, servletoutputstream, queyrdef);
                if(poolconnect == null || queyrdef.m_nQSession == 0)
                {
                    if(queyrdef.m_nQSession == 0)
                    {
                        int i = 1;
                        if(poolconnect != null)
                            i = 0;
                        queyrdef.RowsAffected(servletoutputstream, i);
                        queyrdef.EndClient(servletoutputstream);
                    }
                    return;
                }
            } else
            {
                poolconnect = getPoolConnect(servletoutputstream, queyrdef);
            }
            if(poolconnect == null)
```

```
            return;
        long l = System.currentTimeMillis();
        Calendar calendar = m_ConnLogger != null ? Calendar.getInstance() : null;
        try
        {
            ExecuteQuery(httpservletrequest, poolconnect.m_conn,
servletoutputstream, queyrdef);
        }
        catch(SQLException sqlexception)
        {
            SQLErrMsgSendClient(sqlexception, servletoutputstream, queyrdef);
        }
        catch(ServletException servletexception)
        {
            String s1 = "ServletException From elsjdbc.ExecuteQuery():";
            s1 = s1 + servletexception.getMessage();
            queyrdef.WriteError(s1, servletoutputstream);
        }
        catch(IOException ioexception)
        {
            String s2 = "IOException From elsjdbc.ExecuteQuery():";
            s2 = s2 + ioexception.getMessage();
            queyrdef.WriteError(s2, servletoutputstream);
        }
        catch(Exception exception)
        {
            String s3 = "Exception From elsjdbc.ExecuteQuery():";
            CharArrayWriter chararraywriter = new CharArrayWriter();
            PrintWriter printwriter = new PrintWriter(chararraywriter);
            printwriter.println(s3);
            exception.printStackTrace(printwriter);
            s3 = chararraywriter.toString();
            queyrdef.WriteError(s3, servletoutputstream);
        }
        catch(InternalError internalerror)
        {
            String s4 = "InternalError From elsjdbc.ExecuteQuery():";
            s4 = s4 + internalerror.getMessage();
            queyrdef.WriteError(s4, servletoutputstream);
```

```
            }
            finally
            {
                if(m_ConnLogger != null)
                    WriteLogPool(calendar, new String("RunQuery"));
                if(poolconnect != null)
                    freeConnection(poolconnect, flag, s);
                queyrdef.EndClient(servletoutputstream);
                servletoutputstream.flush();
                servletoutputstream.close();
                LogMsg(l, queyrdef.m_stSQL);
            }
        }

        public void RunQuery(HttpServletRequest httpservletrequest, HttpServletResponse
httpservletresponse, Connection connection)
            throws ServletException, IOException
        {
            QueyrDef queyrdef = new QueyrDef();
            queyrdef.m_nExecCount = 0;
            ServletOutputStream servletoutputstream =
httpservletresponse.getOutputStream();
            if(!GetInPutString(httpservletrequest, servletoutputstream, queyrdef))
                return;
            PoolConnect poolconnect = null;
            boolean flag = false;
            String s = null;
            if(queyrdef.m_nQSession >= 0)
            {
                flag = true;
                if(m_stDriverClass == null)
                {
                    queyrdef.WriteError("Can't Define DriverClass Name",
servletoutputstream);
                    return;
                }
                HttpSession httpsession = httpservletrequest.getSession(true);
                if(httpsession != null)
                    s = httpsession.getId();
```

```
                if(httpsession == null || s == null)
                {
                    queyrdef.WriteError("Can't getSession()", servletoutputstream);
                    return;
                }
                poolconnect = getSessionPoolConnect(s, servletoutputstream, queyrdef);
                if(poolconnect == null || queyrdef.m_nQSession == 0)
                {
                    if(queyrdef.m_nQSession == 0)
                    {
                        int i = 1;
                        if(connection != null)
                            i = 0;
                        queyrdef.RowsAffected(servletoutputstream, i);
                        queyrdef.EndClient(servletoutputstream);
                    }
                    return;
                }
            } else
            if(connection == null)
            {
                queyrdef.WriteError("Can't Connection JDBC driver", servletoutputstream);
                return;
            }
            long l = System.currentTimeMillis();
            try
            {
                if(flag)
                    ExecuteQuery(httpservletrequest, poolconnect.m_conn,
servletoutputstream, queyrdef);
                else
                    ExecuteQuery(httpservletrequest, connection, servletoutputstream,
queyrdef);
            }
            catch(SQLException sqlexception)
            {
                SQLErrMsgSendClient(sqlexception, servletoutputstream, queyrdef);
            }
            catch(ServletException servletexception)
```

```
        {
            String s1 = "ServletException From elsjdbc.ExecuteQuery():";
            s1 = s1 +  servletexception.getMessage();
            queyrdef.WriteError(s1, servletoutputstream);
        }
        catch(IOException ioexception)
        {
            String s2 = "IOException From elsjdbc.ExecuteQuery():";
            s2 = s2 +  ioexception.getMessage();
            queyrdef.WriteError(s2, servletoutputstream);
        }
        catch(InternalError internalerror)
        {
            String s3 = "InternalError From elsjdbc.ExecuteQuery():";
            s3 = s3 +  internalerror.getMessage();
            queyrdef.WriteError(s3, servletoutputstream);
        }
        catch(Exception exception)
        {
            String s4 = "Exception From elsjdbc.ExecuteQuery():";
            CharArrayWriter chararraywriter = new CharArrayWriter();
            PrintWriter printwriter = new PrintWriter(chararraywriter);
            printwriter.println(s4);
            exception.printStackTrace(printwriter);
            s4 = chararraywriter.toString();
            queyrdef.WriteError(s4, servletoutputstream);
        }
        finally
        {
            if(flag && poolconnect != null)
                freeConnection(poolconnect, flag, s);
            queyrdef.EndClient(servletoutputstream);
            servletoutputstream.flush();
            servletoutputstream.close();
            LogMsg(l, queyrdef.m_stSQL);
        }
    }

    private PoolConnect getPoolConnect(ServletOutputStream servletoutputstream,
```

```
QueyrDef queyrdef)
        throws IOException
    {
        boolean flag = true;
        PoolConnect poolconnect = null;
        try
        {
            Calendar calendar = m_ConnLogger != null ? Calendar.getInstance() : null;
            poolconnect = getConnection(m_nTimeOut * 1000);
            if(m_ConnLogger != null)
                WriteLogPool(calendar, new String("getConnection"));
        }
        catch(ClassNotFoundException classnotfoundexception)
        {
            String s = "ClassNotFoundException From elsjdbc.getConnection():";
            s = s +  m_stDriverClass;
            s = s +  "₩nMSG:";
            s = s +  classnotfoundexception.getMessage();
            queyrdef.WriteError(s, servletoutputstream);
            flag = false;
        }
        catch(IOException ioexception)
        {
            String s1 = "IOException From elsjdbc.getConnection():";
            s1 = s1 +  ioexception.getMessage();
            queyrdef.WriteError(s1, servletoutputstream);
            flag = false;
        }
        catch(SQLException sqlexception)
        {
            flag = false;
            SQLErrMsgSendClient(sqlexception, servletoutputstream, queyrdef);
        }
        catch(Exception exception)
        {
            String s2 = "Exception From elsjdbc.getConnection():";
            CharArrayWriter chararraywriter = new CharArrayWriter();
            PrintWriter printwriter = new PrintWriter(chararraywriter);
            printwriter.println(s2);
```

```
                exception.printStackTrace(printwriter);
                s2 = chararraywriter.toString();
                queyrdef.WriteError(s2, servletoutputstream);
                flag = false;
            }
            finally
            {
                if(poolconnect == null || !flag)
                {
                    if(poolconnect != null)
                    {
                        freeConnection(poolconnect, false, new String(""));
                        poolconnect = null;
                    }
                    return null;
                }
            }
            return poolconnect;
        }

        private PoolConnect getSessionPoolConnect(String s, ServletOutputStream
servletoutputstream, QueyrDef queyrdef)
            throws IOException
        {
            boolean flag = true;
            PoolConnect poolconnect = null;
            try
            {
                poolconnect = getSessionConnection(s, queyrdef.m_nQSession,
m_nTimeOut * 1000);
            }
            catch(ClassNotFoundException classnotfoundexception)
            {
                String s1 = "ClassNotFoundException From
elsjdbc.getSessionConnection():";
                s1 = s1 + m_stDriverClass;
                s1 = s1 + "₩nMSG:";
                s1 = s1 + classnotfoundexception.getMessage();
                queyrdef.WriteError(s1, servletoutputstream);
```

```
            flag = false;
        }
        catch(IOException ioexception)
        {
            String s2 = "IOException From elsjdbc.getSessionConnection():";
            s2 = s2 + ioexception.getMessage();
            queyrdef.WriteError(s2, servletoutputstream);
            flag = false;
        }
        catch(SQLException sqlexception)
        {
            flag = false;
            SQLErrMsgSendClient(sqlexception, servletoutputstream, queyrdef);
        }
        catch(Exception exception)
        {
            String s3 = "Exception From elsjdbc.getSessionConnection():";
            CharArrayWriter chararraywriter = new CharArrayWriter();
            PrintWriter printwriter = new PrintWriter(chararraywriter);
            printwriter.println(s3);
            exception.printStackTrace(printwriter);
            s3 = chararraywriter.toString();
            queyrdef.WriteError(s3, servletoutputstream);
            flag = false;
        }
        finally
        {
            if(poolconnect == null || !flag)
            {
                if(poolconnect != null && queyrdef.m_nQSession != 0)
                {
                    freeConnection(poolconnect, true, s);
                    poolconnect = null;
                }
                return null;
            }
        }
        return poolconnect;
    }
```

```
    private void SQLErrMsgSendClient(SQLException sqlexception,
ServletOutputStream servletoutputstream, QueyrDef queyrdef)
        throws IOException
    {
        String s = "SQLError:";
        String s1 = sqlexception.getMessage();
        if(s1 != null)
            s = s + s1;
        String s2 = sqlexception.getSQLState();
        if(s2 != null)
        {
            s = s + "₩nState:";
            s = s + s2;
        }
        queyrdef.WriteError(s, servletoutputstream);
    }

    private void ExecuteQuery(HttpServletRequest httpservletrequest, Connection
connection, ServletOutputStream servletoutputstream, QueyrDef queyrdef)
        throws ServletException, IOException, SQLException
    {
        if(queyrdef.m_nQType == 0 || queyrdef.m_nQType == 1)
            ExecuteBatch(connection, servletoutputstream, queyrdef);
        else
        if(queyrdef.m_nQType == 2)
            ExecuteProcedure(connection, servletoutputstream, queyrdef);
        else
        if(queyrdef.m_nQType == 3)
            ExecuteMqSQL(connection, servletoutputstream, queyrdef);
        else
        if(queyrdef.m_nQType == 4)
            ExecuteOraRefCursor(connection, servletoutputstream, queyrdef);
        else
        if(queyrdef.m_nQType == 64)
            ExecuteMultiSQL(connection, servletoutputstream, queyrdef);
        queyrdef.m_nExecCount++;
    }
```

```
private String GetReplaceString(String s)
{
    if(m_stReplaceOld == null)
        return s;
    int i = s.length();
    int j = m_stReplaceOld.length();
    String s1 = null;
    String s2 = null;
    if(m_bReplaceIgnoreCase)
    {
        s1 = s.toLowerCase();
        s2 = m_stReplaceOld.toLowerCase();
    }
    int k = 0;
    int l = 0;
    String s3 = "";
    do
    {
        int i1 = -1;
        if(m_bReplaceIgnoreCase)
            i1 = s1.indexOf(s2, k);
        else
            i1 = s.indexOf(m_stReplaceOld, k);
        if(i1 < 0)
            break;
        s3 = s3 + s.substring(l, i1);
        if(m_stReplaceNew != null)
            s3 = s3 + m_stReplaceNew;
        l = k = i1 + j;
    } while(true);
    if(i - 1 > l)
        s3 = s3 + s.substring(l);
    return s3;
}

private void LogMsg(long l, String s)
{
    if(m_Logger == null)
    {
```

```
            return;
        } else
        {
            long l1 = System.currentTimeMillis();
            Timestamp timestamp = new Timestamp(l1);
            String s1 = timestamp.toString();
            long l2 = l1 - l;
            s1 = s1 + "(" + Integer.toString((int)l2) + ")";
            m_Logger.println("EZgen=>" + s1 + ": " + s);
            return;
        }
    }

    public void WriteLogPool(Calendar calendar, String s)
    {
        if(m_ConnLogger == null || calendar == null)
        {
            return;
        } else
        {
            Calendar calendar1 = Calendar.getInstance();
            String s1 = "<CheckCount>";
            s1 = s1 + m_nCheckPool;
            s1 = s1 + "<Message>";
            s1 = s1 + s;
            s1 = s1 + "₩r₩n";
            s1 = s1 + "<StartTime>";
            s1 = s1 + getCalendarToString(calendar);
            s1 = s1 + "<EndTime>";
            s1 = s1 + getCalendarToString(calendar1);
            s1 = s1 + "₩r₩n";
            m_ConnLogger.println(s1);
            m_nCheckPool++;
            return;
        }
    }

    public String getCalendarToString(Calendar calendar)
    {
```

```
            int i = calendar.get(1);
            int j = calendar.get(2) + 1;
            int k = calendar.get(5);
            int l = calendar.get(11);
            int i1 = calendar.get(12);
            int j1 = calendar.get(13);
            int k1 = calendar.get(14);
            String s = "" + i;
            String s1 = NumToString(j, 2);
            String s2 = NumToString(k, 2);
            String s3 = NumToString(l, 2);
            String s4 = NumToString(i1, 2);
            String s5 = NumToString(j1, 2);
            String s6 = NumToString(k1, 3);
            String s7 = s;
            s7 = s7 + "-";
            s7 = s7 + s1;
            s7 = s7 + "-";
            s7 = s7 + s2;
            s7 = s7 + " ";
            s7 = s7 + s3;
            s7 = s7 + ":";
            s7 = s7 + s4;
            s7 = s7 + ":";
            s7 = s7 + s5;
            s7 = s7 + " ";
            s7 = s7 + s6;
            return s7;
        }

    }
```

나. DB Connection Pool 기능개선

Client User가 접속하였을 경우 User와 Server는 다음과 같은 Connection Pool을 이용하여 쿼리를 수행하고, 수행한 결과를 서버에서 Servlet 형태로 응답한다.

```
import java.io.*;
import java.sql.*;
import java.util.Vector;
import java.util.Enumeration;

import javax.servlet.*;
import javax.servlet.http.*;

import elsoft.jdbc.*;

public class hnwjdbc extends HttpServlet
{
        static final String m_stDriverClass = "oracle.jdbc.driver.OracleDriver";
        static final String m_stURL = "jdbc.oracle:thin:@<host>:<port1521>:<sid>";
        static final String m_stUID = "UID";
        static final String m_stPWD = "PWD";

        static final int m_nMaxThreads = 10;
        //최대 동시 접속 수를 지정합니다.
        static final int m_nTimeOut = 60;
        //최대 동시 접속 수를 초가시 대기시간(초)

        //Encoding을 위한 멤버 ------------------------------------------
        public final static int IN_NONE = 0;
        public final static int IN_ENG_TO_KOR = 1;
        public final static int IN_KOR_TO_ENG = 2;

        public final static int OUT_NONE = 0;
        public final static int OUT_TO_KOR = 1;
        public final static int OUT_ENG_TO_KOR = 2;  //보통 DB NLS가 영문일때

        static final String m_stEncoding = "KSC5601";
        //Unicode를 Encoding 문자(KSC5601,MS949,...)로 변환합니다.
        static final int m_nInEncoding = IN_NONE;
        //Input String 변환형식.
        static final int m_nOutEncoding = OUT_NONE;
        //Output Stringf 변환 형식.
        static final boolean m_bTrim = false;
        //문자 필드 Trim 사용 여부.
```

```
//Log을 위한 멤버 ------------------------------------------------
//static final int m_nLogType = 2;        //0:하지않음,1:System.out,2:사용자 파일
//static final String m_stLogFile =
//static final int m_nConLogType = 2;   //0:하지않음,1:System.out,2:사용자 파일
//static final String m_stConLogFile =

//Pool을 위한 멤버 -----------------------------------------------
private int m_nMonitorInterval = 600;
//600초(10분) 주기 마다 쓰레드가 돌면서 Connection 의 상태를 검점합니다.
private int m_nPoolTimeout = 3000;
//3000초(50분) 동안 놀고 있는 커넥션이 있다면 reap 쓰레드가 제거합니다.

//elsjdbc를 생성 합니다.(필수 사항) ------------------------------
private elsjdbc m_HnwQuery = new
                                elsjdbc(m_nMonitorInterval,m_nPoolTimeout);

//HttpServlet 기본 함수로 Servlet POST 요청시 호출되는 함수입니다.
public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
{
        request.setCharacterEncoding("euc-kr");
        response.setContentType("text/html;charset=euc-kr");

        m_HnwQuery.RunQuery(request,response);
        //elsjdbc Connect Pooling을 사용하여 쿼리를 실행합니다.
}

//HttpServlet 기본 함수로 Servlet GET 요청시 호출되는 함수입니다.
public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
{
        request.setCharacterEncoding("euc-kr");
        response.setContentType("text/html;charset=euc-kr");

        m_HnwQuery.getStatus(request,response);
      //Connect Pooling에 대한 현재 정보(elsjdbc의 현재 정보)를 모니터링합니다.
}
```

```
        //Servlet 초기화시 호출되는 함수입니다.
        //------>
        public void init(ServletConfig config) throws ServletException
        {
                super.init(config);


m_HnwQuery.init(m_stDriverClass,m_nMaxThreads,m_nTimeOut,m_stURL,m_stUID,m_st
PWD);

m_HnwQuery.UseEncoding(m_nInEncoding,m_nOutEncoding,m_stEncoding,m_bTrim);
        //Encoding 문자를 사용할 경우 지정합니다.

        }

        //Servlet 종료시 호출되는 함수로 Connect된 모든 Pooling를 닫습니다.
-----------------------------
        public void destroy()
        {
                super.destroy();
                m_HnwQuery.release();
        }

}
```

5. 참고자료

- 자바 서블렛 프로그래밍 , 한빛미디어
- ABOUT JDBC , 영진.COM